Lucrarea 1:

Introducere. Operatori şi expresii

A. Exemplu de program C

În prima parte a acestei lucrări ne propunem o familiarizare cu caracteristicile limbajului C. În acest scop prezentăm şi analizăm un program ce execută operaţiile unui calculator de buzunar simplu: adunare, scădere, înmulţire şi împărţire a două numere întregi. Pentru aceasta, în mod repetat, se efectuează următorii paşi:

- se afişează prompter-ul ":"
- se citesc de la tastatura linii de forma:

operand operator operand

-se afişează rezultatul operaţiei.

Programul se opreşte la întâlnirea mărcii **EOF** (End Of File), reprezentată de perechea de caractere CTRL-Z.

```
/*-------------------------------------------------------*/
/* Sa se calculeze rezultatul unor expresii de forma a+b,  */
/* a-b,a*b,a/b unde a si b sunt numere intregi             */
/*                                                        */
/*-------------------------------------------------------*/
#include <stdio.h>
#include <stdlib.h>
#define PROMPT ':'

void main(void)
{
   int a,b;       /*operanzii expresiei   */
   char c;        /* operatorul expresiei */
   int rez;       /* rezultatul expresiei */

   while (putchar(PROMPT), scanf("%d%c%d",&a,&c,&b)!=EOF) {
   /* se selecteaza operatia corespunzatoare */
        switch (c) {
            case '+': rez=a+b; break;
            case '-': rez=a-b; break;
            case '*': rez=a*b; break;
            case '/': if (b==0) { printf("\nImpartire la 0");
                                  exit(1); }
                      rez=a/b; break;
            default:  printf("\nOperatie necunoscuta");
                      exit(1);
           }
        printf("\n%d%c%d=%d",a,c,b,rez);
   }
}
```

Observăm că programul conţine trei linii precedate de caracterul **#**, linii care constituie de fapt directive preprocesor, şi o funcţie **main**. În general, un program C are urmatoarea structură:

directive preprocesor
declaraţii şi definiţii globale
funcţii

Preprocesorul efectuează operaţii anterioare compilării propriu-zise, operaţii ce sunt specificate prin directive preprocesor. Acestea au întotdeauna ca şi prim caracter simbolul **#**. În programul prezentat există două tipuri de directive: **include** şi **define**.

Cele două directive **include** au ca efect includerea în program a fişierului **stdio.h**, respectiv a fişierului **stdlib.h**. Prezenţa acestor directive este impusă de necesitatea declarării funcţiilor de bibliotecă apelate: **putchar()**, **printf()**, **scanf()** (declarate în fişierul **stdio.h**) şi **exit()** (declarată în **stdlib.h**). Astfel, în C, în loc ca programatorul să folosească propriile sale declaraţii pentru funcţiile de bibliotecă, se recomandă includerea în cadrul programului a unor fişiere antet ("header") care conţin declaraţiile funcţiilor de bibliotecă apelate. Aceste fişiere au extensia **.h** şi se găsesc în directorul **INCLUDE**.

Cel de-al doilea tip de directivă preprocesor prezentă în acest program este directiva **define**. Prezenţa ei are ca efect substituirea textuală în cadrul programului a identificatorului *PROMPT* cu caracterul **':'**.

După cele trei directive preprocesor, programul continuă cu funcţia denumită **main**. Orice program C trebuie să conţină o funcţie **main**, execuţia programului începând cu execuţia acestei funcţii. În exemplul prezentat, funcţia **main** nu returnează nici un rezultat (este de tipul **void**) şi are lista de parametri vidă. În cadrul acestei funcţii sunt definite variabile întregi *a* şi *b* - pentru operanzi, şi variabila *rez* - pentru rezultat. De asemenea, este definită variabila caracter *c* - pentru operatorul expresiei.

În continuarea programului, execuţia repetată a paşilor enuntaţi anterior este realizată prin instrucţiunea de ciclare **while**. În cadrul acestui ciclu se apelează funcţia de bibliotecă **putchar**() pentru afişarea caracterului *PROMPT* pe ecran, după care se citesc cu ajutorul funcţiei **scanf**() cei doi operanzi şi operatorul expresiei. Primul parametru al lui **scanf**() este şirul de control al formatului care indică prin specificatorii de format (precedaţi de caracterul **%**) tipul variabilelor din lista de argumente. În cazul nostru avem cei doi operanzi de tip întreg, în forma de reprezentare zecimală, pentru care se foloseşte descriptorul de format **%d**, şi operatorul, care este un caracter specificat prin descriptorul **%c**. În lista de argumente a lui **scanf**() trebuie precizate nu variabilele care se citesc, ci adresele acestora, deci în cazul nostru: *&a, &c, &b*.

Ciclul **while** continuă până la citirea combinaţiei CTRL-Z, situaţie în care funcţia **scanf**() returnează constanta **EOF** (definită în fişierul **stdio.h**).

Funcţie de caracterul citit ca operator se selectează una din cele patru operaţii permise. Această selecţie multiplă este implementată printr-o instrucţiune **switch**, analoagă instrucţiunii **case** din limbajul Pascal. Se observă că la împărţire se testează dacă împărţitorul este nul. În caz afirmativ, se dă un mesaj de eroare, cu ajutorul funcţiei **printf**(), şi se opreşte execuţia programului prin apelul funcţiei **exit**(). În aceeaşi manieră se procedează la citirea unui operator incorect (alternativa **default** a instrucţiunii **switch**).

În cazul unei expresii corecte se afişează expresia, după care caracterul **'='** şi rezultatul expresiei. Pentru aceasta se apelează funcţia **printf**(). Primul parametru al funcţiei este şirul de control al formatului de afişare. El poate conţine caractere care se afişează pe ecran aşa cum apar ele în acest şir (spaţii, mesaje, etc.), precum şi o listă de specificatori de format (prefixaţi de **%**) pentru valorile de afişat. Al doilea parametru al funcţiei este lista (posibil vidă) a valorilor ce se afişează.

B. Probleme propuse

În a doua parte a lucrării prezentăm câteva exerciţii menite să vă familiarizeze cu o parte din operatorii limbajului C.

1. Fiind date definiţiile:

```
int i=3,j=5;
```

determinaţi valoarea următoarelor expresii, precum şi valorile lui *i* şi *j*, după efectuarea celor 4 expresii.

```
a) (i/2) +4
b) (j%3) * i
c) (i++) - (--j)
d)  j = (i += 2)
```

2. Fiind date definiţiile:

```
int a=2, b=2, c=1, d=0, e=4;
```

determinaţi valoarea următoarelor expresii:

```
a) a++ / ++c * --e
b) --b * c++ - a
c) -b - --c
d) ++a - --e
e) e / --a * b++ / c++
f) a%=b=d=1+e/2
```

3. Fiind date definiţiile:

```
int i=2;
    j=4;
```

precizaţi care vor fi valorile lui *i* şi *j* după execuţia următoarei instrucţiuni:

```
j = (i++, i-j);
```

4. Rulaţi următorul program pas cu pas vizualizând valorile variabilelor din program. Justificaţi rezultatele obţinute.

```
void main(void)
{
   int a,b,c; float z;

   a=25000;b=15000;
   c=a+b;
   z=a+b;
   z=(float)a+b;
   c=a/b;
   c=a%b;
   c=a>b;
   z=a/b;
   z=a/(float)b;
   c=a=b;
   c=a!=b;
   a=3;b=11;
   c=a++ + ++b;
   a=0; b=1;
   c=a&&b;
   c=a||b;
   c=!a;
   a=2;b=5;
   c=a-5,b++;
   c*=a+b
   c=(a>b)?a:b;
}
```

A. Consideraţii teoretice

Limbajul C a fost conceput ca un limbaj relativ redus, evitându-se elemente care nu au fost considerate strict necesare sau care ar reduce flexibilitatea şi viteza de execuţie. Prin urmare, C care nu are incorporate facilităţi de prelucrare directă a şirurilor de caractere, mulţimilor, listelor, tablourilor. De asemenea, nu sunt prevăzute în limbaj facilităţi de intrare/ieşire (nu există instrucţiuni READ sau WRITE), de alocare dinamică de memorie sau metode de acces la fişiere. Pentru aceste operaţii, cea mai mare parte a implementărilor de C oferă colecţii standard de funcţii de bibliotecă. Aceste funcţii oferă o varietate de opţiuni. În plus, fiecare programator îşi poate construi propria sa bibliotecă de funcţii care să înlocuiască sau să extindă colecţia de funcţii standard ale limbajului.

Ne propunem în cadrul acestei lucrări să trecem în revistă şi să exemplificăm funcţiile de bibliotecă cele mai uzuale. (Precizăm că funcţiile pentru tratarea şirurilor de caractere şi funcţiile relative la fişire sunt prezentate în lucrarea 4, respectiv lucrarea 5.)

1. Funcţii de intrare-ieşire

a. declarate în stdio.h

getchar

Prototip:

int getchar (void);

Efect: returnează un caracter citit de la tastatură sau constanta simbolică **EOF** dacă s-a întâlnit sfârşitul fişierului de intrare (perechea CTRL-Z).
Observaţie: Funcţia aşteaptă apăsarea tastei **Enter**.

putchar

Prototip:

int putchar (int c);

Efect: afişează caracterul *c* pe ecran; returnează caracterul afişat sau valoarea **EOF** la detecţia unei erori.
gets

Prototip:

char *gets (char *s);

Efect: citeşte un şir de caractere *s* de la tastatură până la întâlnirea caracterului **newline**; returnează adresa şirului citit sau valoarea **NULL** dacă s-a citit CTRL-Z.

puts

Prototip:

int puts (char *s);

Efect: afişează pe ecran şirul *s*.

printf

Prototip:

int printf (char *format, arg1, arg2, ...);

Efect: afişează pe ecran valorile din lista de argumente, conform formatului specificat. Şirul format poate conţine caractere ordinare, care se vor afişa ca atare, şi descriptori de format prefixaţi de caracterul %. Un descriptor de format poate conţine în ordine următoarele:

- un semn minus care indică alinierea la stânga în cadrul formatului a valorii afişate;
- un număr care specifică lungimea minimă a câmpului de afişare;
- un punct care separă lungimea câmpului de afişare de precizia de afişare (de ex. numărul de zecimale pentru valorile reale);

În tabelul următor sunt arătate caracterele de conversie cele mai uzuale ce pot apărea în descriptorii de format:

Caracter	**Valoare**
d,i	întreg zecimal
o	întreg octal fără semn
x, X	întreg hexazecimal fără semn
u	întreg zecimal fără semn
c	caracter
s	şir de caractere
f	real simplă precizie
lf	real dublă precizie

Prezentăm mai jos 3 modalităţi de afişare a şirului de caractere “hello, world”, utilizând diverse formate:

%s :hello, world:
(şirul se afişează pe un câmp egal cu lungimea şirului)
%-15s :hello, world :
(şirul se afişează pe un câmp de lungime 15, aliniat la stânga)
%15s : hello, world:
(se afişează pe un câmp de lungime 15, aliniat, implicit, la dreapta)

scanf

Prototip:

int scanf (char *format, adr1, adr2,...);

Efect: citeşte caractere de la tastatură, conform formatului, şi înscrie valorile citite la adresele specificate. Funcţia returnează numărul de câmpuri citite cu succes sau valoarea **EOF** la întâlnirea sfârşitului fişierului de intrare. Şirul format poate conţine:

- caractere blank sau tab care sunt ignorate;
- caractere ordinare care trebuie să fie citite corespunzător de la intrare;
- specificatori de conversie, la fel ca la **printf**.

Specificatorii de conversie determină modul de conversie a câmpurilor de intrare. Un câmp de intrare se defineşte ca o secvenţă de caractere, altele decât spaţiile albe ale limbajului: blank, tab, newline, carriage return, formfeed, tab vertical.

De exemplu, pentru citirea unei linii de forma:

6 Mar 1996

se va scrie următoarea secvenţă:

```
int zi, an;
char luna[4];

scanf (“%d %s %d”, &zi, luna, &an);
```

Se observă că la variabilele *zi* şi *an* s-a folosit operatorul de adresă &; la variabila *luna* el nu apare deoarece în limbajul C numele unui tablou este echivalent cu adresa sa (este un pointer către primul său element).

Observaţie: Pentru mai multe informaţii privind funcţiile **scanf** şi **printf** citiţi fişierul de help prezent în mediul Turbo C.

sscanf

Prototip:

int sscanf (char *sir,char *format, adr1, adr2,...);

Efect: citeşte date cu formatare; spre deosebire de **scanf** la care citirea se face de la tastatură, la **sscanf** citirea datelor se face dintr-un şir (vezi exemplul 1).

b. declarate în conio.h

getch

Prototip:

int getch (void);

Efect: citeşte un caracter de la tastatură fără să-l afişeze pe ecran; returnează caracterul citit.

getche

Prototip:

int getche (void);

Efect: citeşte un caracter de la tastatură şi-l afişează pe ecran; returnează caracterul citit.

ungetch

Prototip:

int ungetch (int c);

Efect: reinserează caracterul *c* în buffer-ul tastaturii, astfel încât următorul caracter care va fi citit este cel reinserat.

2. Funcţii matematice

Aceste funcţii sunt declarate în fişierul antet **math.h**

Prototipul funcţiei	**Efect**
double acos (double x);	arccosinus de x
double asin (double x);	arcsinus de x
double atan (double x);	arctangentă de x
double atan2 (double y, double x);	arctangentă de y/x
double ceil (double x);	cel mai mic întreg mai mare sau egal cu x
double cos (double x);	cosinus de x
double exp (double x);	exponenţiala e^x
double fabs (double x);	valoarea absolută a lui x
double floor (double x);	cel mai mare întreg mai mic sau egal cu x
double log (double x);	ln de x
double log10 (double x);	lg de x
double pow (double x, double y);	x la puterea y
double sin (double x);	sinus de x
double sqrt (double x);	radicalul lui x

double tan (double x);	tangenta lui x

3. Funcţii de conversie

Prezentăm două funcţii de conversie a unui şir de caractere într-o valoare întreagă, respectiv într-o valoare reală. Aceste funcţii sunt declarate în fişierul antet **stdlib.h**.

Prototip	**Efect**
int atoi (char *s);	converteşte şirul s într-o valoare întregă
double atof (char *s);	converteşte şirul s într-o valoare reală

B. Exemple

1. Să se scrie un program care citeşte o succesiune de linii ce conţin date calendaristice în formatul: **zi/lună/an**. Se cere să se afişeze liniile scrise corect conform formatului specificat anterior.

```
#include <stdio.h>

void main(void){
 char linie[80];
 int zi, luna, an;

 while (gets(linie)!=NULL)
    if(sscanf(linie, "%d/%d/%d", &zi, &luna, &an)==3)
         printf("%s\n", linie);
}
```

2. Să se realizeze programul care citeşte o secvenţă de numere reale, terminată cu **EOF** şi afişează după fiecare citire suma valorilor absolute citite până în acel moment. Afişarea sumei se face cu o precizie de 3 cifre zecimale.

```
#include <stdio.h>
#include <math.h>

void main(void){
 double suma, valoare;
 suma=0;

 while(1) {
   printf("\ndati un numar:");
   if(scanf("%lf", &valoare) == EOF) break;
   printf("\t %.3lf\n", suma+=fabs(valoare));
 }
}
```

3. Se citeşte de la tastatură un text terminat cu **EOF**. Se cere să se afişeze toate numerele întregi conţinute în text.

```
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#define N   15

void main(void){

 char c;    /* caracterul citit la un moment dat */
 char s[20];/*buffer pentru caracterele ce compun un numar */
 int numere[N];      /* tablou cu numerele citite */
 int i=0;
 int j=0;

 c=getchar();
 while (c != EOF)
    if (isdigit(c))  /* incepe un numar */
      {
        i=0;s[i++]=c;
        while (isdigit(c=getchar()))
          s[i++]=c;
        s[i]='\0';
        numere[j++]=atoi(s);
       }
    else
      c=getchar();

 /* afisarea numerelor */
 for (i=0; i<j; i++)
   printf("\n %6d", numere[i]);
}
```

Observaţie:

Pentru a testa dacă un caracter este cifră s-a folosit funcţia **isdigit()** declarată în fişierul antet **ctype.h** (vezi lucrarea 4). După ce s-au citit caracterele corespunzătoare unui număr, folosind funcţia **atoi,** se face conversia în numărul întreg corespunzător. Numerele întâlnite în cadrul textului se memorează în tabloul *numere.*

C. Probleme propuse

1. Scrieţi instrucţiunea care va afişa următoarele linii:

```
H. Smith /78 \ aaa #67 "ggg"
mmm
```

2. Arătaţi ce afişează următorul program:

```
void main(void){

  int i=43;
  char c='A';
```

```
  float f=55.45;

  printf("%5d %.1f %c\n",i,f,c);
  printf("%d %d %d \n",i, (int)f, c);
  printf("%c %d\n", c+1, c);
}
```

3. Să se scrie un program care citeşte de la tastatură un număr real *x* şi un caracter ce reprezintă opţiunea utilizatorului privind operaţia ce trebuie efectuată cu numărul respectiv, după care afişează rezultatul operaţiei. Prezentăm mai jos opţiunile posibile:

'e' - e la puterea ***x***
'l' - ln de ***x***
'r' - radicalul lui ***x***
'm' - valoarea absolută a lui ***x***.

A. Consideraţii teoretice

Spre deosebire de alte limbaje de nivel înalt, de exemplu Pascal Standard, C oferă o familie de 6 operatori la nivel de bit, furnizând astfel facilităţi specifice, de obicei, limbajelor de asamblare. Aceştia permit scrierea unor programe care lucrează îndeaproape cu sistemul hardware al calculatorului.

Operatorii la nivel de bit se aplică unor întregi, adică obiecte de tipul **char**, **short**, **int**, **long**, cu sau fără semn, iar rezultatele obţinute sunt de asemenea întregi. Faţă de ceilalţi operatori ai limbajului, ei operează asupra fiecărui bit din reprezentarea internă a operanzilor, tratându-l independent de valorile celorlaltor biţi.

Aceşti operatori sunt:

~ operator unar de negare
& operatorul binar "şi"
| operatorul binar "sau"
^ operatorul binar "sau exclusiv"
<< operator de deplasare la stânga
>> operator de deplasare la dreapta.

Un exemplu tipic de utilizare a operatorilor logici la nivel de bit este implementarea unei mulţimi ca vector de biţi. În acest caz, un bit dintr-un întreg fără semn se asociază unui element din mulţime, iar valoarea 1, respectiv 0 a unui bit semnifică prezenţa, respectiv absenţa elementului din mulţime. Numărul de elemente din mulţime este limitat la numărul de biţi disponibili. În acest caz operatorul binar & este interpretat ca şi intersecţie, iar | ca şi reuniune.

B. Exemple

1. Pentru exemplificarea operaţiilor la nivel de bit considerăm următoarea secvenţă din fişierul antet **ctype.h**:

```
#define _IS_SP  1           /* is space */
#define _IS_DIG 2           /* is digit indicator */
#define _IS_UPP 4           /* is upper case */
#define _IS_LOW 8           /* is lower case */
#define _IS_HEX 16          /* [0..9]or[A-F] or [a-f] */
#define _IS_CTL 32          /* Control */
#define _IS_PUN 64          /* punctuation */
```

S-a definit un tablou **_ctype[]**, în care caracterului *c* din tabela ASCII îi este asociat elementul de indice *c*+1. Prin acest tablou, fiecarui caracter i se pune în corespondenţă un întreg ce îi specifică tipul.

De ex.:

- caracterul 'A' este un caracter literă mare şi un caracter hexazecimal, deci:

_ctype['A'+1]=0x14 (00010100 în binar, 20 în zecimal)

- caracterul 'm' este un caracter literă mică, deci:

_ctype['m'+1]=0x08 (00001000 în binar, 8 în zecimal)

- caracterul '1' este cifră şi caracter zecimal, deci:

_ctype['0'+1]=0x12 (00010010 în binar, 18 în zecimal)

- caracterul ' ' este caracter spaţiu, deci:

_ctype[' '+1]=0x01 (00000001 în binar, 1 în zecimal)

- caracterul ';' este caracter de punctuaţie, deci:

_ctype[';'+1]=0x40 (01000000 în binar, 64 în zecimal):

Prin urmare, macrodefiniţiile care returnează valoarea 1 dacă caracterul *c* este cifră, literă, respectiv literă sau cifră, sunt implementate astfel:

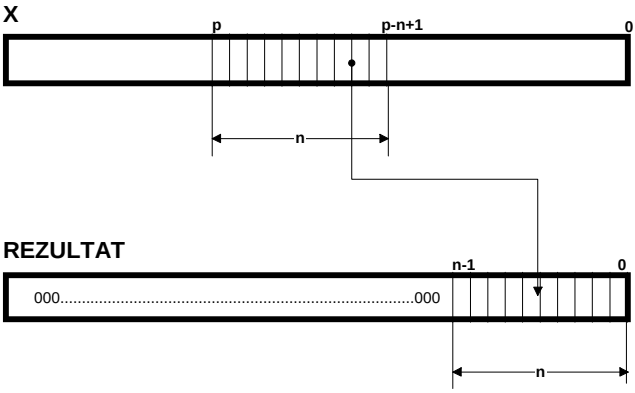
```
#define isdigit(c) (_ctype[(c) + 1] & _IS_DIG)
#define isalpha(c) (_ctype[(c) + 1] & (_IS_UPP|_IS_LOW))
#define isalnum(c) (_ctype[(c)+1]& (_IS_DIG|_IS_UPP|_IS_LOW))
```

În mod analog, sunt implementate şi următoarele macrodefiniţii:

```
#define iscntrl(c)  (_ctype[(c) + 1] & _IS_CTL)
#define islower(c)  (_ctype[(c) + 1] & _IS_LOW)
#define ispunct(c)  (_ctype[(c) + 1] & _IS_PUN)
#define isspace(c)  (_ctype[(c) + 1] & _IS_SP)
#define isupper(c)  (_ctype[(c) + 1] & _IS_UPP)
#define isxdigit(c) (_ctype[(c) + 1] & (_IS_DIG | _IS_HEX))
```

2. Să se scrie o funcţie *getbits(x,p,n)* care returnează, aliniat la dreapta, un câmp de *n* biţi dintr-un întreg fără semn *x*, începând cu poziţia *p* a lui *x*. Se

consideră că bitul cel mai puţin semnificativ al lui *x* are poziţia 0.

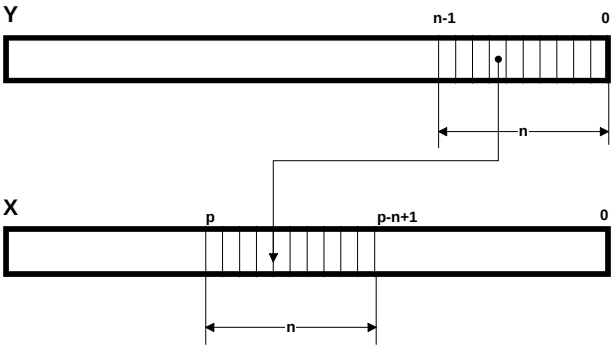



```
#include <stdio.h>

    unsigned getbits( unsigned x, int p, int n){
       return ( x >> (p+1-n) ) & ~(~0 << n);
    }

    void main(void){
      unsigned x;
      int p=6,n=3;

      printf("numar initial:");
      scanf("%u",&x);
      printf("valoarea extrasa: %u\n",getbits(x,p,n));
   }
```

Se observă că în funcţia *getbits*:
- secvenţa x >> (p+1-n) aliniază la dreapta biţii de extras
- secvenţa ~(~0 << n) crează masca de extragere, adica 000...0111

C. Probleme propuse

1. Să se scrie o funcţie *setbits(x,p,n,y)*, care returnează un întreg *x* având *n* biţi începând de la poziţia *p*, egali cu cei mai din dreapta *n* biţi ai întregului *y*. Restul biţilor lui *x* ramân nemodificaţi.



2. Se consideră un întreg DATE_TIME, pe patru octeţi (tipul **long**), în care avem memorată informaţia privind timpul sistemului, astfel:

secundele pe 5 biţi
minutele pe 6 biţi
ora pe 5 biţi
ziua pe 5 biţi
luna pe 4 biţi
anul pe 7 biţi.

Să se scrie un program care oferă în mod repetat următoarele operaţii:

-afişarea datei şi a timpului curent, prin interpretarea variabilei DATE_TIME de tip **long int** (iniţializată la începutul programului cu ajutorul funcţiei *getbits*);

-modificarea unei componente a datei sau a timpului cu ajutorul funcţiei *setbits*;

3. Să se scrie două funcţii *bin(x)* şi *octal(x)* care afişează reprezentarea binară, respectiv octală, a unui întreg *x*, utilizând operatorii de deplasare la nivel de bit.

4. Se consideră mulţimi cu elemente întregi având valori în intervalul [1..n]. Aceste mulţimi se implementează ca vectori de biţi: char tab[n/8+1], în care sunt setaţi biţii corespunzători elementelor prezente în mulţime. Să se implementeze următoarele operaţii (funcţii) cu aceste mulţimi:

- afişarea elementelor unei mulţimi:
```
void afiseaza (char mult[]);
```
- adăugarea unui element la o mulţime:
```
void adauga (int x, char mult[]);
```
- verificarea prezenţei unui element în mulţime:
```
int prezent (int x, char mult[]);
```
- reuniunea a două mulţimi:
```
void reuniune ( char mult1[], char mult2[], char mult3[]);
```

- intersecţia a două mulţimi:
```
void intersectie (char mult1[], char mult2[], char mult3[]);
```
- diferenţa a două mulţimi:
```
void diferenta ( char mult1[], char mult2[], char mult3[]);
```

Se va scrie un program care permite, pe baza unui meniu, selectarea acestor operaţii.

A. Consideraţii teoretice

1. Caractere

Tipul predefinit, destinat în C lucrului cu caractere, este tipul **char**. Spaţiul de memorie asociat variabilelor acestui tip este 1 octet; un caracter se memorează ca numărul întreg corespunzător codului ASCII al caracterului respectiv. Prin urmare, în C un caracter poate apărea în operaţii numerice. Valorile acestui tip sunt cuprinse în intervalul [-128,127] în cazul caracterelor cu semn, respectiv [0,255] în cazul celor fără semn.

Biblioteca C standard oferă o gamă largă de funcţii pentru manipularea caracterelor. Aceste funcţii sunt grupate în două categorii:

- funcţii de citire şi scriere a caracterelor (declarate în fişierul **stdio.h**);
- funcţii de clasificare şi conversie a caracterelor.

O parte din funcţiile din prima categorie au fost tratate în lucrarea 2. În continuare prezentăm câteva funcţii din a doua categorie. La unele dintre acestea vom da şi exemple de implementare.

int isalpha (int c);

Efect: returnează o valoare nenulă dacă *c* este literă ['A'..'Z', 'a'..'z'];

Exemplu:

```
int alpha (int c){
      return (c>='A'&&c<='Z'||c>='a'&&c<='z');
 }
```

int isdigit(int c) ;

Efect: returnează o valoare nenulă dacă *c* este cifră ['0'..'9'];

int isalnum(int c);

Efect: returnează o valoare nenulă dacă *c* este literă sau cifră;

int isupper(int c);

Efect: returnează o valoare nenulă dacă *c* este literă mare;

int islower(int c);

Efect: returnează o valoare nenulă dacă c este literă mică;

int isspace(int c);

Efect: returnează o valoare nenulă dacă *c* este un caracter spaţiu (' ', \r, \t, \n, \f, \v);

int toupper(int c);

Efect: returnează litera majusculă ce corespunde lui *c*, dacă *c* este literă mică;

Exemplu:

```
int toupper(int c){
      return (islower(c)? c-'a'+'A': c);
}
```

int tolower(int c);

Efect: returnează litera mică ce corespunde lui *c*, dacă *c* este literă mare;

2. Şiruri de caractere

În limbajul C nu există prevăzut tipul de date şir de caractere. Pentru memorarea şirurilor se utilizează tablouri cu tipul de bază **char**, care au pe ultima poziţie, ca marcă de sfârşit a şirului, caracterul '\0'.

Pe lânga necesitatea de a defini un şir de caractere ca şi tablou de caractere, în prelucrarea şirurilor se utilizează deseori tipul pointer la caracter. De exemplu:

```
char sir[30];     /*tablou de 30 de caractere */
char * psir;      /* pointer la caracter */
```

Astfel, *sir* este o variabilă de tipul tablou de caractere pentru care se rezervă un spaţiu de memorie de 30 de octeţi, în timp ce *psir* este un pointer la caracter care poate primi ca valoare adresa unui caracter (în particular, adresa primului element dintr-un şir de caractere). Trebuie reţinut că ori de câte ori se lucrează cu variabile şiruri de caractere într-un program, trebuie să existe fie o definiţie de forma celei prezentate pentru variabila *sir*, prin care se rezervă

static (în timpul compilării), spaţiul de memorie necesar variabilei, fie să se aloce dinamic memoria necesară.

Neexistând tipul şir de caractere, în C nu sunt prezenţi nici operatorii pentru şiruri. În schimb, biblioteca standard conţine numeroase funcţii pentru prelucrarea şirurilor de caractere. O parte a funcţiilor pentru citirea/scierea şirurilor, declarate în fişierul antet **stdio.h,** le-am prezentat în lucrarea 2. În continuare, trecem în revistă funcţiile pentru compararea, copierea, concatenarea şirurilor, ş.a.m.d. Acestea sunt declarate în fişierul **string.h**. Ele primesc adresele şirurilor prelucrate prin intermediul parametrilor de tipul pointer la caracter.

int strcmp(char *s1,char *s2);

Efect: compară şirurile *s1* şi *s2* şi returnează <0, daca şirul *s1*< şirul *s2*, 0 dacă *s1* egal *s2*, şi >0 dacă *s1* > *s2*.

Arătăm mai jos două posibilităţi de implementare a acestei funcţii.

```
int strcmp(char *s1,char *s2)
{
 int i;
 for(i=0;s1[i]||s2[i];i++)
   if(s1[i]<s2[i])
             return -1;
          else
             if(s1[i]>s2[i])
                return 1;
 return 0;
}
```

```
int strcmp(char *s1,char *s2)
 {
    int i;
   for(i=0;s1[i]||s2[i];i++)
     if(s1[i]!=s2[i])
                  break;
    return (s1[i]-s2[i]);
   }
```

Observaţie: Expresia de test din ciclurile **for** devine falsă când este atins simultan sfârşitul ambelor şiruri de caractere, prin urmare s1[i] şi s2[i] sunt egale cu '\0', iar s1[i] || s2[i] este 0 (adică valoarea logică fals).

int strncmp(char *s1,char *s2, int n);

Efect: identică cu **strcmp**, dar se compară şirurile *s1* şi *s2* pentru cel mult *n* caractere.

char * strcpy(char *d, char *s);

Efect: copiază şirul sursă *s* în şirul destinaţie *d* şi returnează adresa şirului destinaţie.

Exemplu:

```
char *strcpy( char *d, char *s){
  int i=0;
  while(s[i]){  /* nu s-a ajuns la sfirsitul sirului s */
```

```
    d[i]=s[i];i++;
  }
  d[i]='\0';
  return d;
 }
```

char *strncpy (char *d,char *s, int n);

Efect: copiază maxim *n* caractere de la sursa *s* la destinaţia *d* şi returnează adresa şirului destinaţie.

int strlen(char *s);

Efect: returnează lungimea şirului fără a număra caracterul terminator('\0').

char *strcat(char *d, char *s);

Efect: concatenează cele două şiruri şi returnează adresa şirului rezultat.

Exemplu:

```
char *strcat(char *d, char *s){
  int i=0,j=0;
  while(d[i]) i++;
  while(d[i++]=s[j++]);
  return d;
 }
```

B. Exemple

1. Să se scrie o funcţie care converteşte un caracter reprezentând o cifră hexazecimală în echivalentul întreg zecimal.

```
int hexa (char c){
   if (isalpha(c))
     return toupper(c)-'A'+10;
  return c-'0';
}
```

2. Se citesc, de la intrare, şiruri de caractere, până la întâlnirea caracterului sfârşit de fişier. Să se afişeze de la dreapta la stânga fiecare şir citit.

```
# include <stdio.h>
# include <string.h>

void main(void){
  char sir[80],c;
  short i,j;
  while (gets(sir)!=NULL){
    for(i=0,j=strlen(sir)-1;i<j;++i,--j)
     /* se interschimba prin intermediul lui c elementele
```

```
 aflate pe pozitii simetrice in raport cu mijlocul sirului */
      c=sir[i],sir[i]=sir[j],sir[j]=c;
    puts(sir);
  }
}
```

3. Să se realizeze un program analizor lexical, care preia un text de la tastatură pînă la întâlnirea caracterului de sfârşit de fişier **EOF** şi afişează toate simbolurile întâlnite. Mulţimea simbolurilor recunoscute este:

- identificator (ID) = o succesiune de litere şi cifre, începând cu o literă;
- număr (NR) = o secvenţă de cifre;
- operator (OP) = unul din operatorii <, >, >=, <=, ==, +, -, /, *, =;
- sfârşit de fişier (SF);
- eroare (EROARE) = orice alt caracter diferit de spaţiu;

Spaţiile şi sfârşiturile de linie sunt ignorate. Funcţia **alex()** delimitează următorul simbol din intrare.

```
# include <stdlib.h>
# include <stdio.h>
# include <ctype.h>
# define ID     1
# define NR     2
# define OP     3
# define SF     4
# define EROARE 5

char c;        /* caracterul curent */
char sir[32];  /* buffer pentru identificator */
int val;       /* valoarea unui numar */

/*-------------------------------------------------------------*/
/*                                                             */
/*citeste caractere si determina urmatorul simbol intilnit */
/*                                                             */
/*-------------------------------------------------------------*/

int alex(void){

 int i;
 if(c!=EOF){
   while(isspace(c))  /* se elimina caracterele spatiu */
      c=getchar();
   if (isalpha(c)){  /* incepe un identificator */
     i=0;
     while(isalnum(c)){
       /*  se citesc caracterele dintr-un identificator
           si se memoreaza in sirul s */
       sir[i++]=c;
       c=getchar();
     }
     sir[i]='\0';
     return ID;
   }
   else if (isdigit(c)){ /*incepe un numar */
```

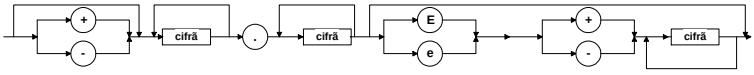
```
        /* se citesc caracterele ce compun numarul
         si se determina valoarea numarului */
        val=c-'0';
        while(isdigit(c=getchar()))
          val=val*10+c-'0';
       return NR;
   }
   else
     switch(c){
     /* se verifica daca este unul din operatori  */
     case '<':case '>': case '=': if((c=getchar())=='=')
                                    c=getchar();
                                  return OP;
     case '+':case '-': case '/': case '*' : c=getchar();
                                             return OP;
    case EOF: c=getchar(); return SF;
     default: c=getchar(); return EROARE;
    } /*switch*/
 } /* if */
 return SF;
}

void main(){

  c=getchar();
  while(1){
    switch(alex()){
      case ID:     printf("identificatorul %s\n",sir);break;
      case NR:     printf("numarul %d\n",val);break;
      case OP:     printf("operator\n");break;
      case EROARE: printf("caracter ilegal\n");break;
      case SF:     printf("sfirsit introducere\n");exit(0);
    }
  }
}
```

C. Probleme propuse

1.Implementaţi următoarele funcţii: **isdigit**, **tolower**, **isupper**, **isalnum**, **strncpy**, **strncmp**, **strlen**.

2. Se dă o listă de cuvinte separate prin spaţiu, listă care se încheie cu caracterul **EOF**. Să se tipărească în ordine alfabetică toate palindroamele care apar în listă. Un palindrom este un cuvânt care este identic citit de la început spre sfârşit ca şi de la sfârşit spre început (de exemplu: *cazac*, *elevele*, *ele*).

3. Un număr real are următoarea diagramă sintactică:



Scrieţi o funcţie care converteşte un şir de caractere, ce respectă această diagramă, în valoarea reală corespunzatoare. Funcţia primeşte ca parametru pointerul la şirul de convertit şi returnează valoarea reală calculată. Funcţia elimină eventualele spaţii ce preced numărul.

Scrieţi un program care afişează suma unei coloane de numere reale, citite ca şi şiruri, câte un număr pe rând. Se va utiliza funcţia definită anterior.

De exemplu:

```
  1.3
  12.01
   +5.7e-1
suma=13.88
```

4. Să se scrie un program care editează un text X într-un text Y. Textul se păstrează sub forma unui tablou de şiruri de caractere. Editarea presupune ştergerea, înlocuirea sau inserţia unor linii. Programul acceptă în mod repetat comenzi de forma:

I,m	Inserţia unei linii de text după linia *m*
D,m,n	Ştergerea liniilor de la *m* la *n*
R,m,n	Înlocuirea liniilor de la *m* la *n*
P,m	Afişarea textului editat începând cu linia *m*
E	Terminarea editării

A. Consideraţii teoretice

În C, spre deosebire de alte limbaje de programare, operaţiile de intrare/ieşire nu se efectuează pe baza unui tip de date şi a unor funcţii incorporate in limbaj. Deoarece aceste prelucrări depind de caracteristicile sistemului de operare, ele sunt furnizate de funcţii din bibioteca standard a limbajului.

Funcţiile utilizate până în acest moment (de ex: **getchar**, **printf**, **gets**, etc.) operează cu echipamentele standard de introducere şi extragere: tastatura şi ecranul monitorul. În cadrul acestei lucrări trecem în revistă câteva funcţii pentru operaţii de intrare/ieşire care transferă date între calculator şi fişierele memorate pe disc.

Prelucrarea unui fişier presupune în primul rând *deschiderea* sa, operaţie prin care i se asociază un canal de intrare/ieşire sau flux (*stream*), se crează o variabilă structură de tipul **FILE** (definit în **stdio.h)** şi se rezervă spaţiu pentru

zona tampon utilizată la transferul datelor. Structura de tipul **FILE** conţine informaţii dependente de implementare cum ar fi: adresa şi lungimea zonei tampon de date, modul de utilizare a fişierului, indicatorul de sfârşit de fişier, indicatorul de poziţie, etc.

În C există două tipuri de stream-uri:

- *text* - care este o succesiune de linii de text despărţite prin caracterul **NL** (newline).

- *binar* - care este o succesiune de octeţi fără nici o structură.

Prezentăm în continuare funcţiile mai uzuale pentru prelucrarea fişierelor.

Deschiderea/închiderea fişierelor

La lansarea în execuţie a unui program se deschid trei fişiere cu canale de tip text:

- **stdin** - pentru intrarea standard, asociat implicit tastaturii;

- **stdout** - pentru ieşirea standard, asociat implicit ecranului monitorului;

- **stderr** - pentru ieşirea standard pentru erori, asociat implicit ecranului monitorului.

Orice alte fişiere utilizate în program trebuie deschise în mod explicit cu funcţia **fopen**.

FILE *fopen(const char *filename, const char *mode);

Prin aceasta se deschide fişierul cu numele *filename* (care poate include şi specificarea căii), i se asociază un stream şi se returnează un pointer către structura **FILE** creată sau **NULL**, dacă fişierul nu a putut fi deschis. Acest pointer este folosit în operaţiile ulterioare asupra fişierului.

Parametrul *mode* este un şir de caractere care poate avea urmatoarele valori:

r deschide pentru citire;

w deschide pentru scriere (dacă fişierul există anterior el va fi şters);

a deschide pentru adăugare (scriere la sfârşitul fişierului);

r+ deschide un fişier existent pentru actualizare (citire şi scriere);

w+ deschide un fişier nou pentru actualizare;

a+ deschide pentru actualizare la sfârşitul fişierului.

La sfârşitul şirului de caractere reprezentând modul de acces se mai poate adăuga opţional sufixul:

t pentru prelucrare în mod text (implicit);

b pentru prelucrare în mod binar.

Închiderea unui fişier deschis cu **fopen** se realizează cu funcţia **fclose**:

int fclose(FILE *fp);

unde *fp* este pointerul returnat de funcţia **fopen**. Prin închiderea fişierului se eliberează şi zona tampon alocată acestuia. Funcţia returnează valoarea 0 dacă operaţia de închidere s-a efectuat cu succes, respectiv **EOF** (-1) în caz de eroare.

Testarea sfârşitului de fişier

Funcţia **feof**:

int feof (FILE *fp);

returnează o valoare nenulă dacă s-a întâlnit sfârşitul de fişier la ultima operaţie de intrare şi 0 în caz contrar.

Golirea buffer-ului unui fişier

Funcţia **fflush**:

int fflush (FILE *fp);

determină golirea buffer-ului unui fişier.

Citirea şi modificarea indicatorului de poziţie

Funcţia **fgetpos**:

int fgetpos (FILE *fp, long int *poz);

înscrie valoarea indicatorului de poziţie în variabila adresată prin pointerul *poz* şi returnează 0 în caz de succes.

Funcţia **setpos**:

int fsetpos (FILE *fp, const long int *poz);

atribuie indicatorului valoarea variabilei indicată prin pointerul *poz* şi returnează valoarea 0 în caz de succes.

Funcţia **fseek**:

int fseek (FILE *fp, long offset, int whence);

repoziţionează indicatorul fişierului la valoarea *whence+offset*; *whence* poate avea urmatoarele valori:

SEEK_SET = 0 - început de fişier
SEEK_CUR = 1 - poziţie curentă
SEEK_END = 2 - sfârşit de fişier

Transferul caracterelor:

Funcţia **getc**:

int getc(FILE *fp);

returnează următorul caracter citit din fişierul *fp* sau **EOF** dacă se întâlneşte sfârşitul fişierului.

Funcţia **ungetc**:

int ungetc(int c, FILE *fp);

repune caracterul *c* în fişierul de intrare *fp*.

Funcţia **putc**:

int putc(int c, FILE *fp);

înscrie caracterul *c* în fişierul *fp*; în caz de eroare returnează **EOF**.

Transferul şirurilor de caractere

Funcţia **fgets**:

char *fgets(char *s, int n, FILE *fp);

citeşte maxim *n*-1 caractere din fişierul *fp* sau până la '\n' inclusiv, le depune în *s*, adăugă la sfârşit '\0' şi returnează adresa şirului *s*; în caz de eroare întoarce valoarea **NULL**.

Funcţia **fputs**:

int fputs(const char *s, FILE *fp);

scrie şirul *s* în fişier, fără caracterul '\0'; în caz de eroare întoarce valoarea **EOF**.

Citirea şi scrierea cu format

Funcţia **fscanf** (similară cu **scanf**):

int fscanf (FILE *fp, const char *format [, adresa, ...]);

citeşte date din fişierul *fp*, conform formatului specificat; returnează numărul câmpurilor de intrare pentru care citirea, conversia şi memorarea s-au efectuat corect sau **EOF** dacă s-a întâlnit sfârşitul fişierului.

Funcţia **fprintf** (similară cu **printf**):

int fprintf (FILE *fp, const char *format [, argum...]);

scrie date, conform formatului în fişierul *fp*; returnează numărul de octeţi transferaţi sau **EOF** în caz de eroare.

B. Exemple

1. Să se scrie un program care afişează lungimea celei mai lungi linii din fişierul text TEST:

```
/*------------------------------------------------------------*/
/*                                                            */
/*       afiseaza lungimea lg_max a celei mai lungi           */
/*               linii a fisierului                           */
/*                                                            */
/*------------------------------------------------------------*/

#include <stdio.h>

void main(void){

 FILE *f;
 char c;
 int lg_max, lg_curenta;

 lg_max=lg_curenta=0;
```

```
 if(!(f=fopen(“TEST”,“r”))){
      puts(“fisierul TEST nu poate fi deschis”);
      return;}

 while ((c=getc(f)) != EOF)
      if (c == ‘\n’) {
             if  (lg_max < lg_curenta)
                 lg_max = lg_curenta;
             lg_curenta = 0;
            }
       else
           lg_curenta++;

 fclose(f);
 printf (“\n linia cea mai lunga are lungimea %d”, lg_max);
}
```

2. Să se scrie un program care copiază un fişier binar sursă în alt fişier binar destinaţie.

```
/*-------------------------------------------------------*/
/*                                                       */
/*         copiaza continutul unui fisier binar          */
/*             in alt fisier binar                       */
/*                                                       */
/*------------------------------------------------------ -*/
#include <stdio.h>

void main(void){
 FILE *fs, *fd;
 char c;
 if ((fs=fopen("sursa", "rb"))==NULL){
    fprintf(stderr, "fisierul sursa nu poate fi deschis \n");
    return;}
 if ((fd=fopen("dest", "wb"))==NULL){
    fprintf(stderr, "fisierul dest nu poate fi deschis \n");
    return;}
 while(1){
      c=getc(fs);
      if (feof(fs))
          break;
      putc(c, fd);
 }
 fclose(fs);
 fclose(fd);
}
```

3. Să se realizeze un program care păstrează evidenţa unei grupe de studenţi. Datele despre studenţi (numele, vârsta, media) se păstrează sub forma unui fişier text. Programul trebuie să permită următoarele opţiuni:

a, A adăugarea unui nou student în fişier;
l, L listarea datelor tuturor studenţilor;
m, M modificarea datelor unui student;
x, X terminarea programului;

```
/*-------------------------------------------------------------*/
/*                                                             */
/*     evidenta unei grupe de studenti                         */
/*                                                             */
/*-------------------------------------------------------------*/


#include <stdio.h>
#include <conio.h>
#include <ctype.h>
#include <stdlib.h>
#include <string.h>
#define LNUME  20          /* lungimea maxima a numelor  */

typedef struct {
       char nume[LNUME];
       int virsta;
       float medie;
     } student;

char fisier[13];          /* numele fisierului */
```

```
/*-------------------------------------------------------------*/
/*                                                             */
/*  adauga datele unui nou student in fisier                   */
/*                                                             */
/*-------------------------------------------------------------*/

void adauga(void){

 FILE *f;
 student s;

 if (!(f=fopen(fisier, "at"))){
       puts("\nfisierul nu poate fi deschis");
       return;
  }
  printf("\n dati numele, virsta, media:");
  scanf("%s %d %f", s.nume, &s.virsta, &s.medie);
  fflush(stdin);        /*se goleste buffer-ul tastaturii */
  fprintf(f, "%20s %2d %6.2f\n",s.nume,s.virsta, s.medie);
  fclose(f);
}
```

```
/*-------------------------------------------------------------*/
/*                                                             */
/*  afiseaza  datele tuturor studentilor                       */
/*                                                             */
/*-------------------------------------------------------------*/
```

```
void listare(void){

 FILE *f;
 student s;

 if (!(f=fopen(fiºier, "rt"))){
       puts("\nfiºierul nu poate fi deschis");
       return;
   }
 while(fscanf(f,"%s %d %f", s.nume,&s.virsta, &s.medie)!=EOF)
    printf("\n%-20s %-2d %-6.2f\n",s.nume,s.virsta, s.medie);
 fclose(f);
}


/*-------------------------------------------------------------*/
/*                                                             */
/*           modifica  datele studentului                      */
/*    al carui nume se citeste in variabila n                  */
/*                                                             */
/*-------------------------------------------------------------*/

void modifica(void){

 int gasit=0;
 FILE *f;
 student s;
 char n[LNUME];  /* numele studentului ale carui date se vor
                    modifica */

 if (!(f=fopen(fiºier, "r+t"))){
       puts("fiºierul nu poate fi deschis");
       return;
   }
  printf("\n numele studentului:");
  gets(n);
  while(fscanf(f,"%s %d %f",s.nume,&s.virsta, &s.medie)!=EOF)
    if(!strcmp(n,s.nume)){
      /* studentul a fost gasit,se afiseaza datele sale */
      printf("%-20s %-2d %-6.2f\n",s.nume,s.virsta, s.medie);
      gasit=1;
      break;
      }
  if (!gasit)
      printf("\n studentul %s nu exista in fiºier", n);
  else {
      printf("\n dati numele, virsta, media:");
      scanf("%s %d %f",s.nume, &s.virsta, &s.medie);
      fflush(stdin);
      /* pozitionare la inceputul inregistrarii */
      fseek(f,-30,SEEK_CUR);
      fprintf(f,"%20s %2d %6.2f\n",s.nume,s.virsta, s.medie);
     }

 fclose(f);
}

/*-------------------------------------------------------------*/
/*                                                             */
```

```
/*   afiseaza  meniul programului                          */
/*                                                         */
/*---------------------------------------------------------*/

void meniu(void){
  puts ("\na, A ---- adaugare student");
  puts ("m, M ---- modificare date student");
  puts ("l, L ---- listare studenti");
  puts ("x, X ---- parasire program");
 }

void main(void){

char opt;

puts("nume fisier:");
gets(fisier);

while(1){
 meniu();
 opt=tolower (getche());

switch(opt){
   case 'a' : adauga();
              break;
   case 'm' : modifica();
              break;
   case 'l' : listare();
              break;
   case 'x' : exit(0);
   default  : puts("comanda eronata\n");
  }
 }
}
```

Se observă că cele trei operaţii principale efectuate asupra fişierului: adăugare, listare şi modificare sunt implementate în trei funcţii distincte.

Funcţia *adauga* înscrie în fişier datele unui student sub forma unei înregistrări de 30 de octeti (20 pentru nume, 1 spaţiu, 2 pentru vârstă, 1 spaţiu, 6 pentru medie). Acest lucru este folosit în funcţia *modifica* când se repoziţionează indicatorul de poziţie în fişier la începutul înregistrării care urmează a fi modificată.

C. Probleme propuse

1. Într-un fişier "cod.dat" este memorată o tabelă de codificare, sub forma:

caracter_de_codificat ' ' caracter_codificat '\n'

Se cere să se preia un text din fişierul "dat1.dat", să se codifice, iar textul obţinut să se scrie în fişierul "dat2.txt".

2. Se consideră două fişiere care conţin numere întregi în ordine strict crescătoare, câte un număr pe o linie. Se cere să se creeze un al treilea fişier, obţinut prin interclasarea primelor două. Dacă un număr apare în ambele fişiere de intrare, el va apărea o singură dată în fişierul rezultat.

De exemplu:

fişier1:
-1 0 4 5 67 123 678 2677
fişier2:
8 12 123 1234 1235 1236 7890 7892
fişier3(rezultat):
-1 0 4 5 8 12 67 123 678 1234 1235 1236 2677 7890 7892

3. Scrieţi un program care concatenează două fişiere, FIS1 şi FIS2 în fişierul FIS3.

4. Dintr-un fişier de intrare se citeşte o propozitie terminată cu punct. Propoziţia este formată din cuvinte ce conţin doar litere mari. Cuvintele sunt separate prin unul sau mai mulţi separatori din mulţimea {' ','\t','\n'}. Ştiind că propoziţia are cel mult 30 de cuvinte, să se afişeze grupurile de cuvinte în care fiecare membru reprezintă anagrama altui membru din grup. Un grup va conţine numărul maxim de membri, iar membrii sunt distincţi.

A. Consideraţii teoretice

1. Structurarea programelor C pe mai multe fişiere

Utilitatea structurării programele C mari pe mai multe fişiere este explicată de următoarele avantaje:

a) Fişierele pot fi compilate independent. Acest aspect este foarte util în faza de corectare a erorilor sintactice şi semantice, când doar fişierul eronat trebuie recompilat. Dacă întregul program este plasat într-un singur fişier atunci orice modificare sau corecţie impune recompilarea întregului program.

b) Structurarea programelor pe mai multe fişiere permite simularea mecanismului de încapsulare a datelor. Variabilele şi funcţiile având clasa de memorare **static** pot fi accesate doar în cadrul fişierului în care ele au fost definite. Variabilele şi funcţiile declarate cu clasa de memorare **extern** sunt

definite într-un alt fişier decât cel curent, dar ele pot fi accesate în fişierul curent. Astfel fişierele apar ca un mecanism de control al vizibilităţii obiectelor.

2. Lucrul în modul “project”

Pentru a compila fişierele şi a le "linkedita" împreună folosim modul **project**. În mediul Borland C acest lucru presupune următorii paşi:

a) Selectăm opţiunea P***roject*** si apoi O***pen project***.

b) Fixăm numele fişierului ***project*** (care are extensia *prj*).

c) Cu ajutorul butonul ***Ins*** precizăm fişierele care trebuie compilate şi apoi "linkeditate" împreună. Cu butonul ***Del*** fişierul selectat este eliminat din grupul celor care sunt compilate şi "linkeditate" împreună.

d) Selectăm opţiunea pentru construirea programului executabil. Mediul BorlandC compilează automat toate fişierele enumerate şi le "linkeditează" împreună.

e) Prin selectarea opţiunii ***Project*** şi apoi a celei ***Close project***, se încheie lucrul în modul ***Project***.

3. Tipuri de date abstracte

Tipul de date abstract este o entitate manipulată doar prin operaţiile ce definesc acel tip. Avantajele utilizării tipurilor de abstracte sunt:

a) Programele devin independente de modul de reprezentare a datelor. Modul de reprezentare poate fi modificat, fără însă a afecta restul programului (de exemplu, o mulţime poate fi implementată printr-un tablou sau printr-o listă ordonată, dar partea de program ce foloseşte operatorii tipului abstract rămâne neschimbată).

b) Se previne violarea accidentală a datelor. Utilizatorul tipului abstract este forţat să manipuleze datele doar prin intermediul operatorilor ce compun tipul abstract, astfel reducându-se riscul unei distrugeri a datelor.

După cum este exemplificat în continuare, în C tipurile abstracte sunt realizate folosind fişiere. Acest mod are desigur propriile sale limitări. Menţionăm doar două din ele:

a) Nu se pot defini tablouri de tipuri abstracte.

b) Nu se pot transmite parametri având ca şi tip un tip abstract.

4. Definirea tipului abstract stivă

Stiva este un tip special de listă în care toate inserţiile şi suprimările de noduri au loc la un singur capăt. Acest capăt se numeşte *vârful* stivei.

Tipul abstract stivă pe care îl definim conţine următorii operatori:

a) Iniţializarea stivei.
b) Verificarea dacă stiva e plină.
c) Verificarea dacă stiva e goală.
d) Introducerea unui element în vârful stivei.
e) Extragererea unui element din vârful stivei.
f) Furnizarea elementul din vârful stivei fără a-l extrage.

În exemplul nostru, stiva este materializată printr-un tablou de numere reale, numit *stiva*, iar vârful stivei este indicat de variabila *ind_top*. Dimensiunea stivei este egală cu *MAX*, valoare care este lungimea tabloului.

Toate declaraţiile şi definiţiile care urmează mai jos trebuie să fie conţinute într-un singur fişier. Pentru a "ascunde" de utilizator detaliile de implementare a stivei, atât *stiva* cât şi *ind_top* sunt definite cu clasa de memorare **static**. Astfel domeniul lor de vizibilitate se restrânge la fişierul curent şi aceste variabile nu pot fi accesate dintr-un alt fişier.

Funcţiile au implicit clasa de memorare *extern*. Prin urmare, ele pot fi apelate din alte fişiere, iar toate operaţiile asupra stivei sunt realizate doar prin intermediul lor.

```
static double stiva[ MAX ];            /* stiva */
static int ind_top;                   /* virful stivei */
```

Operatorii sunt implementaţi astfel:

a) Iniţializarea stivei:

```
 void initializare( void )
{
  int i;

for (i = 0; i < MAX; i++)
     stiva[ i ] = 0; /* toate elementele devin 0 */
ind_top= -1; /* virful stivei indica primul element ocupat*/
} /* initializare */
```

b) Verifică dacă stiva este plină:

```
int plin( void )
 {
```

```
      return ind_top == MAX - 1;

   } /* plina */
```

Stiva este plină atunci când vârful stivei indică spre ultimul element din stivă (cel de indice *MAX - 1*).

c) Verifică dacă stiva este goală:

```
int gol( void )
  {

      return ind_top == -1;

   } /* gol */
```

Funcţia *gol* verifică dacă stiva este goală. Când indicele stivei este egal cu -1, stiva nu conţine nici o valoare şi *gol* returnează 1. Dacă *ind_top* > -1 atunci stiva conţine cel puţin o valoare, astfel că *gol* returnează 0.

d) Introduce un element în vârful stivei:

```
   void push( double nr )
   {

      if( plin() ) /* daca stiva este plina */
      {
        printf("Eroare: stiva este plina\n");
        exit( 1 );
      }
      else
        stiva[ ++ind_top ] = nr; /* noul element este introdus
                                    in virful stivei */

   } /* push */
```

Rutina *push* introduce în vârful stivei numărul transmis prin parametrul *double nr*. Dacă stiva este plină atunci se afişează un mesaj de eroare şi execuţia programului este terminată. În caz contrar, se incrementează vârful stivei *ind_top*, după care noul element este memorat în vârful stivei.

e) Extrage un element din vârful stivei:

```
   void pop( void )
   {

     if( gol() ) /* daca stiva este goala */
     {
       printf("Eroare: stiva este goala\n");
```

```
      exit( 1 );
     }
     else
      ind_top--; /* decrementeaza virful stivei */

   } /* pop */
```

Rutina *pop* extrage elementul din vârful stivei fără a-l returna. Dacă stiva este goală atunci *pop* afişează un mesaj de eroare şi execuţia programului este încheiată. În caz contrar, vârful stivei este decrementat.

f) Furnizează elementul din vârful stivei fără a-l extrage:

```
double top( void )
  {

   if( gol() ) /* daca stiva este goala */
   {
    printf("Eroare: stiva este goala\n");
    exit( 1 );
    }

    else
     return stiva[ ind_top ]; /* returneaza elementul din
                                 virful stivei */
  } /* top */
```

Funcţia *top* returnează valoarea elementului din vârful stivei, însă fără a-l extrage. Dacă stiva este goală atunci se afişează un mesaj de eroare şi execuţia programului este încheiată. În caz contrar, *top* returnează valoarea din vârful stivei.

5. Evaluarea unei expresii în notaţie poloneză

Notaţia poloneză (sau *postfix*) este un mod de scriere a expresiilor, în care ordinea operatorilor şi a operanzilor este schimbată faţă de cea dintr-o expresie uzuală.

În notaţia uzuală (numită şi *infix*) o expresie are forma :

operand operator operand

În notaţia postfix (poloneză) expresia este scrisă sub forma:

operand operand operator

De exemplu:
`a + b` devine în notaţie poloneză `a b +`
`(a - b) * c` devine `a b - c *`
`a - b * (c + d)` devine `a b c d + * -`.

Avantajul notaţiei poloneze este acela că indică ordinea corectă a evaluării operaţiilor fără a utiliza paranteze. Astfel, o expresie poloneză poate fi evaluată printr-o singură parcurgere a sa.

Pentru evaluarea unei expresii în notaţie poloneză se foloseşte o stivă de valori reale. Ori de câte ori întâlnim în expresie un operand, valoarea lui este introdusă în stivă. Dacă întâlnim un operator atunci se extrag două valori din vârful stivei, care sunt, de fapt, cei doi operanzi, aplicăm operatorul asupra valorilor extrase şi rezultatul îl depunem în stivă. În momentul în care s-a terminat parcurgerea expresiei, rezultatul final al expresiei este în vârful stivei (şi stiva conţine doar această valoare).

B. Exemplu

Se citeşte o expresie în notaţie poloneză, care conţine ca operanzi constante reale, iar ca operatori +, -, * şi /. Fiecare expresie apare pe o linie separată. După ce s-a citit expresia, ea este evaluată şi rezultatul ei este tipărit.

Observaţii:

1) Ca stivă se foloseşte tipul abstract stivă cu valori reale, definit anterior.

2) Se defineşte funcţia *getop (char *s)*, care citeşte începând cu poziţia curentă a liniei de intrare şi detectează următorul operand sau operator. *getop* sare peste spaţii şi caracterele tab. Pentru un operand, funcţia returnează constanta *NUMAR* şi, în parametrul *s*, adresa de început a şirului format din cifrele numărului. Pentru un operator, *getop* returnează caracterul citit, iar în parametrul *s* adresa de început a şirului format din acel caracter.

Sursa programului

Programul este împărţit în trei fişiere:

1) *stiva.h* - conţine definiţiile constantelor şi a tipurilor de date folosite în program, precum şi declaraţiile funcţiilor ce prelucrează tipul abstract stivă.

2) *stiva.c* - conţine implementarea tipului abstract stivă.

3) *main.c* - conţine programul principal precum şi funcţia *getop*.

Fişierul *stiva.h*

```
# define  MAX  100 /* dimensiunea stivei */

typedef double stiva_double[ MAX ]; /* tipul stivei */
```

```
/* operatorii care prelucreaza stiva */
void initializare( void );
int plin( void );
int gol( void );
void push( double );
void pop( void );
double top( void );
```

Fişierul *stiva.c*

```
#include <stdio.h>
#include "stiva.h"

 static stiva_double stiva; /* stiva de valori reale */
 static int ind_top;     /* indicatorul virfului de stiva */

 void initializare(void)
 {
     int i;
     for (i = 0; i < MAX; i++)
       stiva[ i ] = 0;
     ind_top = -1;
 } /* initializare */

 int plin(void)
 {
      return ind_top == MAX - 1;
 } /* plin */

 int gol(void)
 {
      return ind_top == -1;
 } /* gol */

 void push(double nr)
 {
      if( plin() )
      {
       printf("Eroare: stiva este plina\n");
       exit(1);
      }
      else
       stiva[ ++ind_top ] = nr;
 } /* push */

 void pop(void)
 {
     if( gol() )
     {
      printf("Eroare: stiva este goala\n");
      exit(1);
     }
     else
      ind_top--;
 } /* pop */

 double top(void)
 {
```

```
     if( gol() )
     {
       printf("Eroare: stiva este goala\n");
       exit(1);
     }
     else
       return stiva[ ind_top ];
 } /* top */
```

Fişierul *main.c*

```
# include <stdio.h>
# include <stdlib.h>
# include <ctype.h>
# include "stiva.h"

# define NUMAR  0   /* constanta returnata de catre getop
                       pentru un operand */
# define M_SIR  30  /* dimensiunea liniei de text */

# define EOLN    -1 /* constanta returnata de getop la
                      sfirsitul unei linii */


int getop (char *s)
{
  int i,c;

  while (( c = getchar()) == ' ' || c == '\t')
       ; /* sare peste spatii si tab */
  s[ 0 ] = c; s[ 1 ] = '\0'; /* salveaza in sirul indicat de
                                s, primul caracter */

  if( !isdigit( c )) /* daca nu e inceputul unui operand */
    if( c!='\n' )
      return c;      /* s-a intilnit un operator si el este
                        returnat*/
    else
      return EOLN;    /* pentru sfirsit de linie */

  i = 0;
  while(isdigit(s[ ++i ]= c = getchar()))
           ; /* operandul este citit in sirul indicat de s */

  if(c == '.')
    while( isdigit( s[ ++i ] = c = getchar())); /* se citeste
                                 partea zecimala a operandului */
  s[ i ] = '\0'; /* sfirsitul sirului de caractere */

  ungetc( c, stdin); /* primul caracter care nu mai apartine
                        de operand este "repus" in intrare */
  return NUMAR;

} /* getop */


void main()
{
```

```
  int oper;
  char sir[ M_SIR ];
  double operand1, operand2;

  initializare(); /* sterge continutul stivei*/
  while(( oper = getop( sir )) != EOLN) /* pina cind se
                                citeste toata linia */
    switch( oper )
    {
      case NUMAR: push( atof( sir )); /* se plaseaza
                                operandul in stiva */
                  break;
      case  '+' : operand1 = top(); pop(); /* se extrage
                                     primul operand */
                  operand2 = top(); pop(); /* se extrage al
                                          doilea operand */
                  push(operand1 + operand2); /* suma lor este
                                       introdusa in stiva */
                  break;

      case  '-' : operand1 = top(); pop(); /* se extrage
                                primul operand */
                  operand2 = top(); pop(); /* se extrage al
                                          doilea operand */
                  push(operand2 - operand1); /* diferenta lor
                                   este introdusa in stiva */
                  break;

      case  '*' : operand1 = top(); pop(); /* se extrage
                                          primul operand */
                  operand2 = top(); pop(); /* se extrage al
                                          doilea operand */
                  push(operand1 * operand2); /* produsul lor
                                    este introdus in stiva */
                  break;

      case  '/' : operand1 = top(); pop(); /* se extrage
                                          primul operand */
                  operand2 = top(); pop(); /* se extrage al
                                          doilea operand */
                  if(operand1 == 0)
                   {
                      printf("Eroare: impartire cu 0\n");
                     exit(1);
                    }
                   else
                     push(operand2 / operand1); /* rezultatul
                         impartirii este introdus in stiva */
                    break;

    default: printf("Eroare:operator %c necunoscut\n", oper);
             exit(1);
    } /* switch */

  printf("Valoarea expresiei este %lf\n", top());
  /* rezultatul expresiei este in virful stivei */

} /* main */
```

Comentarea programului

Funcţia *main* citeşte o linie care conţine o expresie în notaţie poloneză, o evaluează şi tipăreşte rezultatul ei. Primul pas iniţializează stiva. În continuare, se preiau operanzii şi operatorii expresiei (apelând funcţia *getop*) până când *getop* returnează *EOLN*. Operanzii sunt introduşi în stivă prin *push*. Dacă este întâlnit un operator, atunci se apelează *top* şi *pop* pentru a citi şi elimina cei doi operanzi din vârful stivei. Se efectuează operaţia aritmetică şi rezultatul ei este introdus în stivă. În cazul împărţirii, se verifică dacă împărţitorul este nenul. După procesarea întregii expresii, rezultatul, care se găseşte în vârful stivei, este afişat.

getop citeşte linia de text şi găseşte operanzii şi operatorii din expresie. *getop* returnează *NUMAR* pentru un operand, *EOLN* pentru sfârşitul liniei sau caracterul corespunzător operatorului găsit. *getop* "sare" peste spaţii şi caracterele tab. Dacă găseşte o cifră înseamnă că a detectat începutul unui operand pe care îl preia şi îl memorează în tabloul indicat prin parametrul *char *s*. Primul caracter ce urmează numărului, este repus în tamponul de intrare astfel încât următorul apel al lui *getop* porneşte chiar cu acest caracter.

C. Probleme propuse

1. Un editor de texte prelucrează câte o linie de text folosind o stivă. Editorul citeşte pe rând câte un caracter. Dacă caracterul citit nu este '<', '>' sau '\n' atunci el este introdus în stivă. Dacă caracterul citit este '<', atunci se şterge un caracter din vârful stivei. Pentru '>' se goleşte stiva. La întâlnirea caracterului '\n' linia editată este afişată.

2. Să se definească tipul abstract tabelă de simboluri, care conţine următorii operatori:

a) introduce în tabelă un identificator împreună cu informaţiile asociate lui;
b) regăseşte în tabelă informaţiile asociate unui identificator;
c) verifică dacă tabela este plină sau nu;
d) iniţializează tabela de simboluri;
e) caută un identificator în tabela de simboluri;
f) afişează identificatorii din tabelă împreună cu informaţiile asociate lor;
g) şterge toţi identificatorii din tabelă;
h) ordonează identificatorii din tabelă.

Tabela poate conţine maxim 200 de identificatori. Informaţia asociată unui identificator se referă la faptul că acesta reprezintă un nume de variabilă, un nume de funcţie, un cuvânt cheie sau o etichetă.

Un identificator :

. urmat de parantezele rotunde () este considerat nume de funcţie.

. urmat de două puncte : este interpretat ca etichetă.

. ce se găseşte într-o tabelă de cuvinte cheie, care se numeşte *key_tab*, este interpretat ca şi cuvânt cheie.

. în celelalte situaţii este considerat variabilă.

Folosind tipul abstract astfel definit să se realizeze următorul program:

Se citeşte un text care se încheie cu **EOF**. Programul găseşte toţi identificatorii din text împreună cu informaţiile referitoare la felul lor. Aceşti identificatori se introduc în tabela de simboluri, asigurând faptul că un identificator apare o singură dată. În caz că un identificator apare de mai multe ori se semnalează un mesaj de eroare şi se păstrează doar prima apariţie a identificatorului. După preluarea tuturor identificatorilor din text se afişează tabela de simboluri. În continuare, tabela este ordonată alfabetic funcţie de identificatori şi tipărită.

3. Să se realizeze un program C ce ţine evidenţa personalului unei companii de dimensiuni mici (aproximativ 50 de angajaţi). Informaţia referitoare la angajaţi este păstrată într-un fişier şi este folosită pentru iniţializarea bazei de date. Fişierul conţine linii de forma:

nume vârsta adresa număr_matricol funcţie

Câmpurile sunt separate printr-un spaţiu şi sunt şiruri de caractere cu următoarele lungimi: nume - 19, vârsta - 2, adresa - 11, număr_matricol - 5 şi funcţie - 5.

Funcţiile care trebuie implementate sunt:

. *initialize_db(db_file)* - iniţializează baza de date din fişierul *db_file*.

. *print_db()* - afişează baza de date.

. *emp_virsta(n)* - returnează vârsta persoanei cu numele *n*.

. *emp_adresa(n)* - returnează adresa persoanei cu numele *n*.

. *emp_numar_matricol(n)* - returnează numărul matricol al persoanei cu numele *n*.

. *emp_functie(n)* - returnează funcţia persoanei cu numele *n*.

. *introdu(n, v, a, nm, f)* - introduce în baza de date o persoană şi datele aferente ei.

. *cauta(n)* - verifică prezenţa unei persoane în baza de date.

. *save_db(db_file)* - salvează baza de date în fişierul *db_file*.

. *delete(n)* - şterge din baza de date persoana cu numele *n*.

. *modif(n)* - permite modificarea informaţiilor legate de persoana cu numele *n*.

A. Consideraţii teoretice

Una dintre caracteristicile de bază ale limbajului C este posibilitatea de a lucra, prin intermediul pointerilor, cu adrese de date şi de funcţii, inclusiv de a efectua operaţii aritmetice şi comparaţii. Aceste caracteristici sunt rezervate, în general, limbajelor de asamblare; înglobându-le, însă, într-un limbaj de nivel înalt, aceste operaţii pot fi efectuate în cadrul unor programe portabile, independente de structura hardware a calculatorului.

Utilizarea pointerilor se impune uneori ca unică alternativă posibilă; alteori, ei conduc la un cod mai compact şi mai eficient decît alte metode. Pointerii se folosesc în C cu precădere în următorele trei situaţii:

- transmiterea de rezultate prin intermediul parametrilor funcţiilor;
- accesarea elementelor tablourilor;
- descrierea şi accesarea structurilor de date dinamice.

Pe lângă pointerii de date care conţin adresa unei variabile sau constante, mai există o a doua categorie - pointerii de funcţii care conţin adresa codului executabil al unei funcţii.

Vom exemplifica, în secţiunea următoare, utilizarea pointerilor de date şi de funcţii în diferite situaţii.

B. Exemple

1. Să se scrie un program care afişează meniul de la exemplul din lucrarea 5, utilizând în acest scop câte o funcţie distinctă pentru fiecare opţiune. Apelarea acestora se face indirect, prin intermediul unui tablou de pointeri, corespunzători adreselor funcţiilor.

```
#include <stdio.h>

typedef void (*pfunc) (); /* pointer la o functie de tipul
                             void,fara parametrii */

void ad(void){
  puts ("\n adauga student");
}

void mod(void){
  puts ("\n modifica date student");
```

```
}

void list(void){
  puts ("\n listeaza datele tuturor studentilor");
}

void term(void){
  puts ("\n terminare program");
}

pfunc tab[]={ad,mod,list,term};

void meniu(void){
   int i=0;
   for (; i<4; i++)
      (*tab[i])();
 }

 void main(void){
   meniu();

 }
```

2. Să se scrie un program care gestionează date despre un grup de studenţi. Pentru fiecare student se memorează numele şi numărul său matricol. Programul trebuie să implementeze următoarele operaţii:

- citirea numărului de studenţi şi a datelor acestora;
- afişarea datelor tuturor studenţilor;
- sortarea listei de studenţi în ordinea alfabetică a numelor;
- sortarea listei de studenţi în ordinea crescătoare a numerelor matricole;
- căutarea unui student pentru care se precizează numele şi afişarea poziţiei pe care o ocupă acesta în lista ordonată alfabetic după numele studenţilor;
- căutarea unui student pentru care se precizează numărul matricol şi afişarea poziţiei pe care o ocupă acesta în lista ordonată crescător după numărul matricol al studenţilor;

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <ctype.h>
#include <conio.h>

typedef struct{
      char *nume;
      int nr;
   }student;

#define SIZE sizeof(student)

typedef int (*comp)(void*, void*); /*tip pointer la functia
                                    de comparare*/
```

```
typedef student * pstud;

 /*-------------------------------------------------------*/
 /*                                                       */
 /* functia eroare afiseaza un mesaj de eroare,in         */
 /* cazul in care nu poate fi alocata dinamic o zona      */
 /* de memorie; apoi se opreste executia programului      */
 /*                                                       */
 /*-------------------------------------------------------*/


 void eroare(void){
    puts(" \n **** eroare alocare dinamica de memorie ****");
    exit(1);
 }

 /*-------------------------------------------------------*/
 /*                                                       */
 /* citeste numarul de studenti si datele acestora,       */
 /* aloca dinamic spatiu pentru tabloul de studenti,      */
 /* memoreaza numele si nr. matricol ale studentilor      */
 /*                                                       */
 /*-------------------------------------------------------*/

 void citeste(int *n, pstud *tab){
    pstud t;
    char sir[40];
    int i;

    /* citeste numarul de studenti */
    printf("\n dati numarul de studenti:");
    scanf("%d", n);

     /*aloca spatiu pentru toate inregistrarile de tip
     student */
    if(!(t=(pstud)malloc ((*n)*SIZE))) /*citeste1*/
       eroare();
    *tab=t;
    /* citeste datele tuturor studentilor */
    for(i=0;i< *n; i++,t++) {
        printf("\n nume: ");
        scanf("%s", sir);
        /*aloca dinamic spatiu pentru numele studentului */
        if(!(t->nume=(char*)malloc(strlen(sir)+1)))
              /*citeste2*/
            eroare();
        strcpy(t->nume,sir);
        printf("\n numar matricol:");
        scanf("%d", &t->nr);
     }
   }
 /*-------------------------------------------------------*/
 /*                                                       */
 /* afiseaza numele si numarul matricol ale celor         */
 /* n studenti din tabelul a carui adresa este data       */
 /* in parametrul tab                                     */
 /*                                                       */
 /*-------------------------------------------------------*/
```

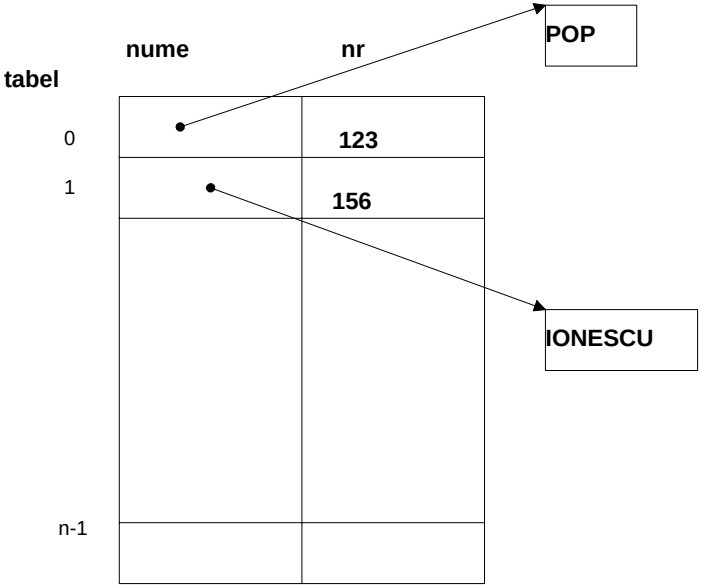
```
 void afiseaza (int n, pstud tab){
 int i;

 puts("\n tabelul cu studenti ");
 for(i=0; i<n; i++, tab++)
    printf("\n%-30s %4d", tab->nume, tab->nr);
 }

 /*-------------------------------------------------------*/
 /*                                                       */
 /* functia comp1 compara numele a doi studenti ale caror  */
 /* adrese sunt date de pointerii p si r                  */
 /*                                                       */
 /*-------------------------------------------------------*/

 int comp1 (void *p, void *r){

   return strcmp (((pstud)p)->nume, ((pstud)r)->nume);

}

 /*-------------------------------------------------------*/
 /*                                                       */
 /* functia comp2 compara nr.matricol a doi studenti ale   */
 /* caror adrese sunt date de pointerii p si r            */
 /*                                                       */
 /*-------------------------------------------------------*/

int comp2 (void *p, void *r){

   return (((pstud)p)->nr - ((pstud)r)->nr);

 }


 /*-------------------------------------------------------*/
 /*                                                       */
 /* functia cauta studentul cu adresa data de parametrul s */
 /* in tabelul cu adresa indicata de parametrul tab;       */
 /* tabelul este ordonat dupa criteriul specificat prin    */
 /*  parametrul f                                         */
 /*                                                       */
 /*-------------------------------------------------------*/

void cauta (pstud s, pstud tab,int n, comp f){
 pstud t=NULL;
 int i;

 t=bsearch((void*)s,(void *)tab,n, SIZE,f);
 if(t){
            /* exista studentul cautat */
            i=t-tab;
            printf ("\n studentul %s cu numarul matricol %d e
al %d -lea in evidenta", t->nume,t->nr,i+1);
          }
   else
             printf("\n studentul nu se afla in evidenta");
  }
```

```
/*-----------------------------------------------------------*/
/*                                                           */
/* elibereaza spatiul de memorie detinut de cimpurile nume*/
/* ale  celor n studenti din tabel si apoi zona detinuta  */
/* de tabel                                                  */
/*                                                           */
/*-----------------------------------------------------------*/

void elibereaza(pstud tabel,int n){
 int i;

 for(i=0; i<n; i++)
   free(tabel[i].nume);
 free(tabel);
}

/*-----------------------------------------------------------*/
/*                                                           */
/* functia meniu  afiseaza meniul programului                */
/*                                                           */
/*-----------------------------------------------------------*/

void meniu(void){
  puts("\n c, C --- citeste tabel studenti");
  puts(" a, A --- afiseaza tabel studenti");
  puts(" n, N --- ordoneaza dupa nume");
  puts(" r, R --- ordoneaza dupa numar matricol");
  puts(" f, F --- cauta dupa nume");
  puts(" l, L --- cauta dupa numar matricol");
  puts(" x, X --- iesire din program");
}


void main(void){

 char opt;
 int n;     /* numarul de studenti */
 char nume[30];
 student s;
 pstud tabel=NULL; /* adresa tabloului cu studenti */


while(1){
  meniu();
  opt=tolower(getche());
  switch(opt){
    case 'c': if(tabel) /* daca a existat anterior un alt
                        tablou in memorie */
                   elibereaza(tabel,n);
              citeste(&n, &tabel);
              break;
    case 'a': afiseaza(n,tabel);
              break;
    case 'n': qsort((void*)tabel, n, SIZE, comp1);
              break;
    case 'r': qsort((void*)tabel, n, SIZE, comp2);
              break;
    case 'f': printf("\n dati numele:");
```

```
                scanf("%s", nume);
                if(!(s.nume=(char *)malloc (strlen(nume)+1)))
                    eroare();
                strcpy(s.nume,nume);
                cauta(&s,tabel,n,comp1);
                free(s.nume);  /*elibereaza spatiul alocat
                                 pentru nume */
                break;
     case 'l': printf("\n dati numarul matricol:");
                scanf("%d", &s.nr);
                cauta(&s,tabel,n,comp2);
                break;
     case 'x': exit(0);
     default:  puts("comanda gresita");
   }
 }
}
```

Observaţii:

1. Din considerente de economie de memorie, implementarea tabloului cu studenţi s-a făcut dinamic; după ce s-a citit numărul de studenţi, s-a alocat spaţiul corespunzător (vezi linia /*citeste1*/). În plus, în tablou, în câmpul *nume* se păstrează nu şirul de caractere pentru numele studenţilor, ci un pointer la şirul de caractere. Astfel, în loc de a rezerva, pentru toate numele, acelaşi număr de caractere, în momentul în care se cunoaşte exact numele unui student, se alocă dinamic un spaţiu de memorie de dimensiune egală cu lungimea acestuia (vezi linia /*citeste2*/). Adresa acestui spaţiu se păstreză în câmpul *nume.*



2. Reamintim că în limbajul C, mecanismul de transmitere a parametrilor este prin valoare. Prin urmare, o funcţie nu poate modifica valoarea unui parametru actual, chiar dacă modifică parametrul formal corespondent. Pentru situaţiile în care se doreşte să se transmită informaţie de la funcţia apelată la cea apelantă prin intermediul parametrilor se procedează în felul urmator: la apelul funcţiei se transmite adresa obiectului care urmează să se modifice, iar, în cadrul funcţiei, parametrul corespondent se declară de tipul pointer. Astfel, functia apelată cunoaşte adresa obiectului respectiv şi îl poate modifica, operând indirect asupra lui, prin intermediul pointerului.

Din aceste considerente, deoarece citirea numărului de studenţi şi alocarea spaţiului pentru tablou se face în funcţia *citeste*, parametrii formali corespunzători *int *n* şi *pstud *tab* vor fi de tipul pointer. La apelul funcţiei *citeste* se vor transmite nu valorile lui *n* şi *tabel*, ci adresele acestora: *&n, &tabel.*

3. La accesarea elementelor tabloului de studenţi s-a folosit atât lucrul cu pointeri în funcţiile *citeste* şi *afiseaza*, cât şi varianta cu indici în funcţia *elibereaza*.

4. Pentru compararea, după nume, a doi studenţi s-a implementat funcţia *comp1*. Similar, funcţia *comp2* realizează compararea după numărul matricol.

5. Pentru sortarea tabloului se apelează funcţia de bibliotecă **qsort**. Aceasta are următorul prototip:

void qsort (void *tab, int nr_elem, int nr_oct_elem, int (*fcmp)(void *, void *));

Parametrii indică adresa tabloului de sortat, numărul elementelor tabloului, mărimea în octeţi a fiecărui element şi adresa funcţiei de comparare.

6. După sortarea tabloului, căutarea unui student se face cu funcţia de căutare binară **bsearch**:

void *bsearch (void *cheie, void *tab, int nr_elem, int nr_octeti_elem, int (*fcmp)(void *,void *));

Parametrii indică cheia căutată, adresa tabloului, numărul elementelor tabloului, mărimea în octeţi a fiecărui element şi adresa funcţiei de comparare. Funcţia returnează adresa primului element care corespunde cheii, respectiv valoarea **NULL** dacă nu există un astfel de element.

7. În funcţia *cauta*, pentru a afla numărul de ordine al unui student în tabelă, se scade din adresa elementului de tablou corespunzător acestuia (*t*), adresa de început a tabelei (*tab*).

8. Pentru ca programul să funcţioneze corect, comenzile *f* şi *l* trebuie date după ce în prealabil s-au dat comenzile *n* şi, respectiv, *r*. (Se utilizează pentru căutarea unui student o funcţie de căutare binară, deci tabela trebuie să fie anterior sortată!)

C. Probleme propuse

1. Precizaţi ce va afişa următorul program; verificaţi-vă rulând programul.

```
#include <stdio.h>

void main(void){
 int t[4] = {0,1,2,3};
 int *p=&t[1];

 printf ("%d\n", *p++);
 printf ("%d\n", *++p);
 printf ("%d\n", ++*p);
}
```

2. Cunoscând că **sizeof (char)** este 1, **sizeof (int)** este 2 şi **sizeof (float)** este 4, determinaţi care sunt valorile obţinute prin atribuirile de mai jos:

```
char *pc;               /* presupunem pc=900 */
int *pi1, *pi2;         /* presupunem pi1=1000 si pi2=1012 */
float *pf1, *pf2;       /* presupunem pf1=1100 si pf2=1112 */
```

```
pc+=6;
pi1+=6;
pf1+=6;
*pf1=pi1-pi2;
*pi1=pf1-pf2;
pi1-=*pi2;
```

3. Să se realizeze un program pentru evidenţa abonaţilor telefonici dintr-o localitate. Pentru fiecare abonat se memorează numele, adresa, numărul de telefon. Programul trebuie să permită:
- adăugarea unui nou abonat;
- ştergerea unui abonat;
- modificarea datelor unui abonat;
- afişarea numărului de telefon al unui abonat.
- ştergerea întregii evidenţe.

Implementarea listei de abonaţi să se facă folosind cât mai puţină memorie.

A. Problemă

Să se realizeze un program care răspunde la următoarele comenzi:

'a' - Se citeşte o linie de forma:
identificator număr
unde *număr* este un număr întreg. Ca rezultat, se reţine în evidenţă identificatorul împreună cu numărul asociat lui.

't' - Se citeşte o linie ce conţine un identificator. Dacă acesta apare în evidenţă, se tipăreşte valoarea asociată lui; în caz contrar se tipăreşte un mesaj de eroare.

's' - Se citeşte o linie ce conţine un identificator şi îl şterge din evidenţă.

'l' - Se tipăresc identificatorii din evidenţă în ordine alfabetică împreună cu valorile asociate lor.

'+' - Se citesc două linii, fiecare conţinând un identificator. În cazul în care ambii se află în evidenţă se tipăreşte suma lor. În caz că unul sau ambii identificatori lipsesc din evidenţă, se afişează un mesaj de eroare.

'-' - Se citesc două linii, fiecare conţinând un identificator. În cazul în care ambii se află în evidenţă se tipăreşte diferenţa lor. În caz că unul sau ambii identificatori lipsesc din evidenţă, se afişează un mesaj de eroare.

'*' - Se citesc două linii, fiecare conţinând un identificator. În cazul în care ambii se află în evidenţă se tipăreşte produsul lor. În caz că unul sau ambii identificatori lipsesc din evidenţă se afişează un mesaj de eroare.

'/' - Se citesc două linii, fiecare conţinând un identificator. În cazul în care ambii se află în evidenţă se tipăreşte rezultatul împărţirii lor. În caz că unul sau ambii identificatori lipsesc din evidenţă se afişează un mesaj de eroare.

'f' - Se termină programul.

Observaţie:
Identificatorii sunt păstraţi în tabelă în mod ordonat.

Sursa programului

```
# include <stdio.h>
# include <stdlib.h>
# include <string.h>
#define  Max   10   /* lungimea maxima a identificatorilor */

typedef struct  elem     /* tipul pentru nodul listei */
  {
          char *id;
          int valoare;
          struct elem  *urm;
   } nod;

nod *radacina = NULL;     /* pastreaza inceputul listei */

/*-------------------------------------------------------------*/
/*                                                             */
/* Functia cautare cauta sirul s in lista al carei inceput */
/* e indicat de parametrul lista. Daca sirul apare in lista*/
/* atunci functia returneaza pointerul la nodul respectiv, */
/*           in caz contrar returneaza NULL                    */
/*                                                              */
/*-------------------------------------------------------------*/
nod *cautare(nod *lista , char *s)
{
 nod *q1 ;

 for(q1 = lista; q1!= NULL && strcmp( q1->id,s ) < 0;
   q1 = q1->urm); /* cautare 1 */
 if(q1 != NULL && strcmp(q1->id,s) == 0) /* cautare 2 */
   /* daca sirul s a fost gasit in lista */
    return q1;
 else return NULL;
}

/*-------------------------------------------------------------*/
/*                                                             */
/*   Functia listare parcurge lista si pentru fiecare nod  */
/*afiseaza identificatorul memorat si valoarea atasata lui.*/
```

```
/*Deoarece  lista este ordonata, afisarea identificatorilor*/
/*               este in ordine alfabetica                  */
/*                                                           */
/*-----------------------------------------------------------*/
void listare(void)
{
 nod *q;

 for( q = radacina ; q != NULL ; q = q->urm )
   printf("identificator: %s valoare %d \n",q->id,
   q->valoare);
}

/*-----------------------------------------------------------*/
/*                                                           */
/* Functia stergere elimina din lista indicata de pointerul*/
/*  lista, nodul ce are cimpul id egal cu argumentul s     */
/*                                                           */
/*-----------------------------------------------------------*/

nod *stergere(nod *lista, char *s)
{

 nod *q1 ,  *q2;

 for(q1 = q2 = lista; q1 != NULL && strcmp(q1->id, s) < 0;
          q2 = q1, q1 = q1->urm); /* stergere 1 */
 /* se parcurge lista cautindu-se nodul avind cimpul id egal
    cu sirul s */

 if(q1 != NULL && strcmp(q1->id, s) == 0) /* stergere 2 */
  {
   /* daca s-a gasit un astfel de nod */
   if(q1 != q2) /* daca nodul nu este la inceputul listei */
         /* stergere 3 */
      q2->urm = q1->urm; /* elimina nodul din lista */
   else /* nodul apare la inceputul listei */
      lista = lista->urm; /* stergere 4 */

   free (q1->id); /* se elibereaza memoria ocupata de nodul
                     eliminat */
   free (q1);

   return lista; /* returneaza pointerul catre inceputul
                    listei modificate */
  }
 else
  {
   printf("Eroare:identificatorul %s nu apare in lista\n",s);
   return lista;
  }
 }


/*-----------------------------------------------------------*/
/*                                                           */
/* Functia introducere insereaza un nod  in lista ordonata,*/
/* indicata de parametrul lista. Lista ramine ordonata si  */
/* dupa inserare. Nodul nou are cimpul id egal cu sirul    */
```

```
/* indicat de parametrul s, iar cimpul valoare egal cu     */
/* parametrul v. Functia returneaza pointerul catre        */
/* inceputul listei modificate                              */
/*                                                           */
/*-----------------------------------------------------------*/

nod *introducere(nod *lista, char *s, int v)
{
 nod *q1, *q2, *aux;
 if((aux = (nod*)malloc(sizeof(nod))) == NULL ||
    (aux->id = (char *)malloc(strlen(s) + 1)) == NULL)
      /* introducere 1 */
      /*daca nu e memorie suficienta pentru a crea un nod
       nou, respectiv pentru a memora sirul s */
  {
   /*se da un mesaj de eroare dupa care executia e
    incheiata*/
   printf(" Eroare: memorie insuficienta\n");
   exit(1);
  }
 else
  {
   strcpy(aux->id,s); /* se salveaza s in nodul nou */
   aux->valoare = v; /* se salveaza v in nodul nou */
  }

 /* nodul nou este inserat in lista ordonata astfel incit ea
ramine  ordonata  si  dupa  inserare.  Lista  este  parcursa
cautindu-se primul nod avind cimpul id mai mare sau egal cu
s*/

 for(q1 = q2 = lista; q1 != NULL && strcmp(q1->id, s) < 0;
        q2 = q1, q1 = q1->urm); /* introducere 2 */

 if(q1 != NULL && strcmp(q1->id, s) == 0) /* introducere 3 */
  /* daca in lista apare un nod avind câmpul id egal cu s,
  atunci se afiseaza un mesaj de eroare si se pastreaza lista
  nemodificata */
   {
     printf("Eroare: %s apare in tabela\n", s);
     return lista;
   }
  else
     if(q1 != q2) /* daca inserarea nu se face la inceputul
                  listei */ /* introducere 4 */
       {
        q2->urm = aux ;
        aux->urm = q1;
        return lista ;
       }
     else /* daca inserarea se face la inceputul listei */
                /* introducere 5 */

       {
        aux->urm = lista;
        return aux;
       }
}
```

```
/*-------------------------------------------------------------*/
/*                                                             */
/*  Functia getlin citeste urmatoarea linie de la tastatura*/
/* si recunoaste identificatorul si valoarea asociata lui. */
/* Identificatorul este returnat in parametrul s,          */
/* iar valoarea in parametrul val                          */
/*                                                              */
/*-------------------------------------------------------------*/
void getlin(char *s ,int *val)
{
 int i = 0, j;
 char temp[Max];

 gets(temp); /* citeste urmatoarea linie de la tastatura */
             /* getlin 1 */
 s[i] = '\0';*val = 0;/*initializeaza valorile argumentelor*/
 while(temp[i] != '\0') /* atita timp cit nu a fost atins
                           sfirsitul sirului */
                        /* getlin 2 */
   if(isalpha(temp[i])) /* daca incepe un identificator */
                        /* getlin 3 */
    {
     j = 0;
     while(isalnum(temp[i])) s[j++] = temp[i++]; /*getlin 4*/
      /* memoreaza identificatorul in s */
      s[j] = '\0'; /* memoreaza sfirsitul de sir */
                   /* getlin 5 */
    }
   else
     if(isdigit(temp[i])) /* daca incepe un numar */
                                /* getlin 6 */
       while(isdigit(temp[i])) /* getlin 7 */
         {
          /* calculeaza valoarea numarului */
          *val = *val * 10 + temp[i] - '0' ; /* getlin 8 */
          i++;
         }
     else i++; /* altfel se trece peste caracterul curent */
               /* getlin 9 */
}


/*-------------------------------------------------------------*/
/*                                                             */
/*Functia comanda_a realizeaza functionalitatea comenzii a */
/*                                                             */
/*-------------------------------------------------------------*/

void comanda_a(void)
{
 int val;
 char s[Max];

 getlin(s, &val); /* citeste o linie de la tastatura */
 if(strlen(s) != 0) /* daca linia e corecta */
    radacina = introducere(radacina, s, val);
```

```
 else printf(" Eroare : linie incorecta \n");
}


/*-------------------------------------------------------------*/
/*                                                             */
/* Functia comanda_t realizeaza functionalitatea comenzii t*/
/*                                                              */
/*-------------------------------------------------------------*/

void comanda_t(void)
{
 char s[Max];
 nod *p;
 int val;

 getlin(s, &val); /* citeste o linie de la tastatura */
 if(strlen(s) != 0) /* daca linia e corecta */
 {
  if((p = cautare (radacina, s)) != NULL) /* cauta nodul in
                                             lista */
   printf("Identificator:%s Valoare:%d \n",p->id,p->valoare);
  else
   printf("Eroare: Identificator nedefinit \n");
  }
 else
  printf(" Eroare: linie incorecta \n");
}


/*-------------------------------------------------------------*/
/*                                                             */
/*        Functia comanda_s realizeaza comanda s               */
/*                                                             */
/*-------------------------------------------------------------*/

void comanda_s(void)
{
 char s[Max];
 int val;

 getlin(s, &val); /* citeste o linie de la tastatura */
 if(strlen(s) != 0)  /*daca linia citita e corecta */
  radacina = stergere(radacina, s);/*sterge nodul din lista*/
 else printf(" Eroare: linie incorecta \n");
}







/*-------------------------------------------------------------*/
/*                                                             */
/*   Functia comanda_oper executa operatiile legate de         */
/*   comenzile +, -, *, /                                      */
/*  Se citesc cei doi operatori si se executa operatia         */
/*  dorita. Rezultatul este afisat.                            */
```

```
/*                                                                  */
/*------------------------------------------------------------------*/

void comanda_oper(char c)
{
 char s1[ Max ], s2[ Max ];
 int val;
 nod *p1, *p2;

 getlin(s1, &val); /* se citeste primul operand */
                   /* comanda_oper 1 */
 getlin(s2, &val); /* se citeste al doilea operand */
                   /* comanda_oper 2 */

 if((strlen(s1) != 0) && (strlen(s2) != 0))
   if(((p1 = cautare (radacina, s1)) != NULL) &&
       ((p2 = cautare (radacina, s2)) != NULL))
        /* comanda_oper 3 */
        /* se verifica daca operanzii apar in lista */

        {
          switch(c) /* functie de tipul comenzii */
           {
             case '+' : val = p1->valoare + p2->valoare;
                        break;
             case '-' : val = p1->valoare - p2->valoare;
                        break;
             case '*' : val = p1->valoare * p2->valoare;
                        break;
              case '/' : if(p2-> valoare != 0)
                           val = p1->valoare / p2->valoare;
                         else
                           printf("Eroare:Impartire la 0\n");
                         break;
           }
           printf(" Rezultatul operatiei e %d \n", val);
        }
   else printf("Operand nedefinit \n");
 else printf(" Eroare: linie eronata \n");
}




/*------------------------------------------------------------------*/
/*                                                                  */
/*Functia meniu afiseaza meniul programului,citeste comanda*/
/* si apeleaza subrutina corespunzatoare                           */
/*                                                                   */
/*------------------------------------------------------------------*/

void meniu(void)
{
 char o;

 while( 1 ) /* meniu 1 */
```

```
 {
  clrscr();
  /* se afiseaza meniul programului */
  printf("a ... adauga in evidenta un id si val asociata\n");
  printf("t ... tipareste valoarea asociata unui id \n");
  printf("s ... sterge un id \n");
  printf("l ... listeaza id-rii si valorile asociate lor\n");
  printf("+ ... calculeaza suma pentru 2 id \n");
  printf("- ... calculeaza diferenta pentru 2 id \n");
  printf("* ... calculeaza produsul pentru 2 id \n");
  printf("/ ... calculeaza impartirea pentru 2 id \n");
  printf("f ... termina programul \n");
  printf("\n Introduceti optiunea :");
  o = getchar(); getchar(); /* meniu 2 */
  switch(tolower(o)) /* meniu 3 */
   {
     case 'a': comanda_a(); break;
     case 't': comanda_t(); break;
     case 's': comanda_s(); break;
     case 'l': listare(); break;
     case '+': case '-': case '*': case '/':comanda_oper(o);
                                            break;
     case 'f' : return;
     default : printf(" Eroare : Comanda inexistenta \n");
   }
  getchar();
 }
}


/*-------------------------------------------------------------*/
/*                                                             */
/*       Programul principal apeleaza functia meniu            */
/*                                                              */
/*-------------------------------------------------------------*/

void main(void)
{
  meniu();
}
```

Comentarea programului

Programul memorează identificatorii şi valorile asociate lor într-o evidenţă, pe care o foloseşte conform funcţionalităţii cerute. Evidenţa este realizată printr-o listă înlănţuită ordonată. Fiecare nod al listei conţine un câmp pentru păstrarea identificatorului, care este indicat de *char *id*, un câmp pentru memorarea valorii asociate identificatorului, câmpul *int valoare*, şi un câmp pentru a crea înlănţuirea cu nodul următor din listă, câmpul *struct elem *urm*. Lista este ordonată crescător funcţie de câmpurile *id* ale nodurilor.

Programul principal apelează funcţia *meniu*. Aceasta afişează în mod repetat, datorită instrucţiunii *while (1)* de pe linia */* meniu 1 */*, opţiunile pe care le oferă programul. În continuare, *meniu* citeşte de la tastatură opţiunea

dorită (linia */* meniu 2 */*) şi caracterul <**Return**>. Funcţie de opţiunea aleasă (linia */*meniu 3 */*) se selectează rutina ce implementează funcţionalitatea dorită.

Rutina *comanda_a* citeşte un identificator şi valoarea asociată lui, după care îl introduce în evidenţă. Citirea se face apelând funcţia *getlin*. În cazul în care lungimea identificatorului citit este 0, se afişează un mesaj de eroare. În caz contrar, identificatorul şi valoarea asociată lui sunt introduse în lista ordonată prin apelarea rutinei *introducere*.

comanda_t citeşte un identificator folosind *getlin*. Dacă lungimea identificatorului este 0, atunci se afişează mesajul că linia este incorectă. Dacă însă lungimea este diferită de 0, atunci identificatorul este căutat în listă (apelând funcţia *cauta*). Dacă identificatorul apare în listă, atunci se afişează valoarea asociată lui, în caz contrar se tipăreşte un mesaj de eroare adecvat.

Funcţia *comanda_s* citeşte o linie ce conţine un identificator (folosind *getlin*). Dacă linia citită este corectă (adică lungimea identificatorului este nenulă), atunci identificatorul este eliminat din evidenţă prin apelarea lui *stergere*.

comanda_oper citeşte doi identificatori (liniile */* comanda_oper 1 */* şi */* comanda_oper 2 */*) şi efectuează o operaţie aritmetică cu aceşti operanzi. Felul operaţiei dorite este transmis prin parametrul *char c* al rutinei *comanda_oper*. Dacă identificatorii sunt corecţi (au lungimile diferite de 0), atunci ei sunt căutaţi în evidenţă (linia */* comanda_oper 3*/*). Dacă ambii apar, atunci, funcţie de *c*, se efectuează operaţia dorită, iar rezultatul este afişat.

Citirea unei linii de la tastatură se face prin rutina *getlin*. Linia */* getlin 1*/* citeşte în variabila locală *temp* o linie de la tastatură. Ciclul *while* de pe linia */* getlin 2 */* parcurge tabloul *temp*. Dacă găseşte o literă (linia */* getlin 3 */*), atunci înseamnă că a descoperit începutul unui identificator. Ciclul *while* de pe linia */* getlin 4 */* preia întregul identificator şi îl memorează în tabloul al cărui început este indicat de parametrul *char *s*. Linia */*getlin 5 */* memorează caracterul '\0' ce indică sfârşitul identificatorului. Dacă la parcurgerea lui *temp* este întâlnită o cifră (linia */* getlin 6 */*), atunci a fost descoperit începutul numărului care apare în linia de text. Acest număr este preluat prin ciclul *while* de pe linia */* getlin 7 */*, calculându-se simultan şi valoarea numerică asociată lui. Linia */* getlin 8 */* face conversia, din şirul de caractere asociat numărului întreg, în valoarea lui. Calcularea valorii numerice se face folosind variabila întreagă a cărei adresă este transmisă prin parametrul *int *val*. Linia */*getlin 9*/* "sare" peste acele caractere care nu fac parte din identificator sau număr.

cautare parcurge lista al cărei început este transmis prin parametrul *nod *lista*, căutând nodul al cărui identificator este egal cu şirul indicat de

parametrul *char *s*. Dacă un astfel de nod este găsit, atunci *cautare* returnează pointerul către el. În caz contrar, returneaza **NULL**. Parcurgerea listei se face prin ciclul *for* de pe linia */* cautare 1 */*. Variabila *q1* este iniţializată astfel încât să indice începutul listei. La sfârşitul fiecărei iteraţii, *q1* este mutat către următorul nod. Condiţia de reluare a ciclului este ca lista să nu fi fost parcursă în întregime (condiţia *q1 != NULL* de pe linia */* cautare 1 */)* şi identificatorul să nu fi fost găsit. Deoarece lista este ordonată crescător este suficient să testăm că *strcmp(q1->id,s) < 0* (linia */* cautare 1 */*). Linia */* cautare 2 */* verifică dacă identificatorul a fost găsit. Pentru aceasta este necesar ca *q1 != NULL* (altfel s-ar fi parcurs toată lista) şi şirul memorat în nodul curent să fie egal cu cel indicat de *s*.

Rutina *listare* parcurge toate nodurile din listă şi, pentru fiecare, afişează identificatorul reţinut şi valoarea ataşată lui.

Rutina *stergere* elimină, din lista indicată de parametrul *nod *lista*, nodul al cărui identificator este egal cu şirul indicat de parametrul *char *s*. *stergere* returnează pointerul către începutul listei modificate. Dupa eliminare, lista rămâne ordonată. Pentru eliminarea unui nod sunt folosiţi doi pointeri care parcurg lista. *q1* indică nodul curent, în timp ce *q2* pe cel anterior lui. Acest lucru este necesar deoarece la eliminarea nodului indicat de *q1*, înlănţuirea *urm* a nodului anterior (care este indicat de *q2*) se modifică spre nodul următor lui *q1*. Ciclul *for* de pe linia */* stergere 1 */* parcurge lista. Pointerii *q1* şi *q2* sunt iniţializaţi să indice spre începutul listei din care se face ştergerea. La fiecare reluare a ciclului, *q2* ia vechea valoare a lui *q1*, în timp ce *q1* se mută spre nodul următor. Condiţia de reluare a ciclului *for* este ca lista să nu fi fost parcursă în întregime (*q1 != NULL*) şi nodul să nu fi fost găsit (*strcmp(q1->id, s) < 0)*. Dacă nodul a fost găsit (linia */* stergere 2 */*), atunci se verifică dacă el este sau nu primul nod al listei. Dacă nodul nu este chiar începutul listei (linia */* stergere 3 */*), atunci înlănţuirea *urm* a nodului anterior celui indicat de *q1* se modifică spre nodul următor celui indicat de *q1*. Dacă însă nodul indicat de *q1* este primul din listă, atunci variabila *lista* este schimbată spre al doilea nod. Memoria ocupată de nodul eliminat este eliberată. Dacă însă nu există un nod care să aibă identificatorul egal cu sirul indicat de *s*, atunci *stergere* tipăreşte un mesaj de eroare.

introducere crează un nod nou care conţine şirul indicat de parametrul *char *s*, iar câmpul lui de valoare este egal cu parametrul *int v*. Nodul este inserat în lista indicată de parametrul *nod *lista*, astfel încât după introducere, lista rămâne ordonată. *introducere* returnează începutul listei modificate. Linia */* introducere 1 */* alocă memorie pentru noul nod. Dacă alocarea nu este posibilă, se afişează un mesaj de eroare şi execuţia programului este încheiată. În caz contrar, câmpurile *id* şi *valoare* sunt iniţializate corespunzator. Linia */* introducere 2 */* parcurge lista căutând poziţia în care noul nod trebuie introdus. Pentru introducerea unui nod într-o listă ordonată sunt necesari doi pointeri; unul indică nodul în faţa căruia se face inserarea, iar al doilea pe cel

anterior lui. Dacă noul nod nu este inserat ca primul nod în listă (linia /* *introducere 4 */*), atunci înlănţuirea de la nodul anterior (indicat de *q2)* se schimbă către nodul nou, iar înlănţuirea noului nod se schimbă spre nodul indicat de *q1*. Dacă nodul devine primul în listă (linia /* *introducere 5 */*) atunci înlănţuirea lui *urm* este modificată spre nodul indicat de *q1*. În acest caz valoarea returnată de *introducere* (care este începutul listei modificate) este tocmai pointerul spre nodul recent introdus. Dacă în listă apare deja un nod al cărui identificator este egal cu cel care se doreşte a fi introdus (linia /* *introducere 3 */*), atunci se afişează un mesaj de eroare şi se păstrează lista nemodificată.

B. Problemă propusă

1. Să se implementeze un set de funcţii care să realizeze următoarele operaţii:

- *adauga (t,id)* - introduce identificatorul *id* în tabela de simboluri *t*;

- *prezent (t,id)* - returnează 1 sau 0 după cum identificatorul *id* este sau nu prezent în tabela *t*;

- *sterge (t,id)* - elimină *id* din tabela *t*;

- *reuniune (t1,t2,t)* - *t* va conţine identificatorii prezenţi în *t1* sau *t2*;

- *intersectie (t1,t2,t)* - *t* va conţine identificatorii prezenţi atât în *t1* cât şi în *t2*;

- *scadere (t1,t2,t)* - *t* va conţine identificatorii prezenţi în *t1* şi absenţi în *t2*;

- *tipareste(t)* - tipăreşte în ordine alfabetică identificatorii din *t.*

Folosind funcţiile de mai sus să se realizeze un program care citeşte două secvenţe consecutive de text, care se termină fiecare cu **EOF**. După citirea textelor se cere să se tipărească, în ordine alfabetică, identificatorii prezenţi doar în primul text , apoi cei doar în al doilea text. În continuare se vor tipări în ordine alfabetică identificatorii care sunt prezenţi atât în primul cât şi în al doilea text. În final se afişează identificatorii care apar în cel puţin unul din cele două texte.

A. Problemă

Să se realizeze un program care răspunde la următoarele comenzi:

'a' - Se citesc trei linii ce conţin datele asociate unei persoane. Prima linie conţine numele, a doua locul de muncă, iar a treia vârsta. Persoana şi datele asociate ei sunt reţinute într-o evidenţă. Dacă persoana este deja prezentă atunci se actualizează datele asociate ei.

't' - Se citeşte o linie ce conţine un nume. În cazul în care acesta este prezent în evidenţă, se tipăresc datele asociate lui. În caz contrar, se afişează un mesaj de atenţionare.

'l' - Se citeşte o linie ce conţine un loc de muncă şi se tipăresc în ordine alfabetică numele tuturor persoanelor ce au locul de muncă respectiv.

'p' - Se citeşte o valoare întreagă şi se tipăresc, în ordinea crescătoare a vârstei, numele şi datele asociate tuturor persoanelor care au vârsta mai mare sau egală cu valoarea citită.

's' - Se citeşte o linie ce conţine un nume. Persoana cu numele respectiv este eliminată din evidenţă.

'd' - Se citeşte o linie ce conţine un loc de muncă şi se elimină din evidenţă toate persoanele care au locul de muncă respectiv.

'n' - Se tipăresc în ordine alfabetică toate persoanele din evidenţă şi datele asociate lor.

'v' - Se tipăresc în ordine descrescătoare a vârstei persoanelor din evidenţă.

'f' - Se termină programului.

Sursa programului

Fişierul *tip.h* :

```
/* tipul unui nod al listei ce implementeaza evidenta */
typedef struct nod_lista
{
  char *nume; /* nume */
  char *loc_munca; /* locul de munca */
  int virsta; /* virsta */
  struct nod_lista *alf;/*inlantuirea în ordine alfabetica */
```

```
  struct nod_lista *vircr; /* inlantuirea în ordinea
                              crescatoarea a virstei */
  struct nod_lista *virdes; /* inlantuirea în ordinea
                               descrescatoarea a virstei */
} nod;

/* operatiile asupra listei */
 nod *cauta( char* );
 void introdu( char*, char*, int );
 void afis_alf( void );
 void afis_des( void );
 void afis_loc_munca( char* );
 void afis_virsta( int );
 void elimin( char* );
 void elimin_loc_munca( char* );
```

Fişierul *lista.c*

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include "tip.h"

static nod *inc1 =NULL, *inc2 =NULL, *inc3=NULL ;
/* nodurile listei sunt înlănţuite dupa cele trei criterii de
ordonare */
/* cele trei liste sunt indicate de inc1, inc2 si inc3 */
/* inc1 indica lista ordonata în ordine alfabetica */
/* inc2 indica lista ordonata în ordinea crescatoare a virstei
*/
/*  inc3  indica  lista  ordonata  în  ordinea  descrecatoare  a
virstei */

static nod *auxp; /* variabila auxiliara folosita la
                     eliminarea unui nod */




/*-------------------------------------------------------------*/
/*                                                             */
/*    cauta în lista ordonata alfabetic nodul care are         */
/*    numele egal cu parametrul n;                             */
/*    returneaza pointerul la nod sau NULL                     */
/*                                                             */
/*-------------------------------------------------------------*/

 nod *cauta( char *n)
 {
  nod *l;

  for(l = inc1;l!=NULL && strcmp(l->nume, n) < 0; l = l->alf)
       ; /* parcurge lista cautind nodul cu nume egal cu n */
  if( l != NULL && strcmp(l->nume,n ) == 0)
     /* daca nodul a fost gasit */
      return l;
```

```
  return NULL;
 } /* cauta */


/*-------------------------------------------------------------*/
/*                                                             */
/*      scot1 elimina din lista ordonata alfabetic nodul       */
/*     care  are numele egal cu parametrul l                   */
/*                                                             */
/*-------------------------------------------------------------*/

 nod *scot1( nod *p, char *l)
   /* p este inceputul listei din care se face eliminarea */
 {
   nod *l1, *l2;

    for(l1 = l2 = p; l1 != NULL && strcmp(l1->nume, l) < 0;
      l2 = l1, l1 = l1->alf)
           ; /* se parcurge lista ordonata alfabetic cautind
             nodul cu numele egal cu l */
    if( l1 != NULL && strcmp(l1->nume, l) == 0)
        /* daca s-a gasit nodul */
        if(l1 == l2 ) /* daca este primul nod în lista */
          return p->alf;
        else
          {
          l2->alf = l1->alf;
          return p;
          }
    else /* nodul nu a fost gasit în lista */
     {
      printf("Eroare: %s nu apare în evidenta\n",l);
      return p;
     }
 } /* scot1 */


/*-------------------------------------------------------------*/
/*                                                             */
/*     scot2 elimina din lista ordonata crescator functie      */
/*  de varsta, nodul care are numele egal cu parametrul l      */
/*                                                             */
/*-------------------------------------------------------------*/

 nod *scot2(nod *p, char *l)
  /* p este inceputul listei din care se face eliminarea */
 {
   nod *l1, *l2;

    for(l1 = l2 = p; l1 != NULL && strcmp( l1->nume,l) != 0;
     l2 = l1, l1=l1->vircr);
      /* se parcurge lista ordonata crescator functie de
      varsta cautind nodul cu numele egal cu l */

    if( l1 != NULL && strcmp(l1->nume, l) ==0)
     /* daca s-a gasit nodul */
      if(l1 == l2 ) /* daca este primul nod în lista */
        return p->vircr;
      else
```

```
       {
         l2->vircr = l1->vircr;
         return p;
       }
    else
     {
      printf("Eroare: %s nu apare în evidenta\n",l);
      return p;
     }
 } /* scot2 */

/*-------------------------------------------------------------*/
/*                                                             */
/*   scot3 elimina din lista ordonata descrescator functie */
/* de varsta nodul care are numele egal cu parametrul l    */
/*                                                             */
/*-------------------------------------------------------------*/
 nod *scot3(nod *p, char *l)
  /* p este inceputul listei din care se face eliminarea */
 {
   nod *l1, *l2;

    for(l1 = l2 = p; l1 != NULL && strcmp(l1->nume, l) != 0;
      l2 = l1, l1 = l1->virdes) ;
    /* se parcurge lista ordonata descrescator functie de
    varsta cautind nodul cu numele egal cu l */
    if( l1 != NULL && strcmp (l1->nume, l) == 0)
     /* daca s-a gasit nodul */
     {
      auxp = l1; /* auxp indica spre nodul care se elimina */
      if(l1 == l2 ) /* daca este primul nod în lista */
          return p->virdes;
      else
      {
        l2->virdes = l1->virdes;
        return p;
      }
    }
    else
    {
      printf("Eroare: %s nu apare în evidenta\n",l);
      return p;
    }
 } /* scot3 */


/*-------------------------------------------------------------*/
/*                                                             */
/*      intr1 introduce nodul indicat de l în lista        */
/*      ordonata alfabetic indicata de p                   */
/*                                                             */
/*-------------------------------------------------------------*/

 nod *intr1(nod *p, nod *l)
  /* p este inceputul listei in care se face introducerea */
  /* l este nodul care este inserat */
  {
   nod *l1,*l2;
```

```
   for(l1 = l2 = p;l1!=NULL && strcmp(l1->nume, l->nume) < 0;
     l2 = l1, l1 = l1->alf) ;
   l->alf = l1;
   if(l1 == l2 )
     /* daca nodul este inserat la inceputul listei */
     return l;
   else
     {
      l2->alf = l;
      return p;
     }
 } /* intr1 */


/*-------------------------------------------------------------*/
/*                                                             */
/*   intr2 introduce nodul indicat de l în lista ordonata  */
/*   crescator functie de varsta, si indicata de p         */
/*                                                             */
/*-------------------------------------------------------------*/

nod *intr2( nod *p, nod *l)
  /* p este inceputul listei in care se face introducerea */
  /* l este nodul care este inserat */
 {
   nod *l1, *l2;

   for(l1 = l2 = p; l1 != NULL && l1->virsta < l->virsta ;
     l2 = l1, l1 = l1->vircr) ;
   l->vircr = l1;
   if(l1 == l2)
      /* daca nodul este inserat la inceputul listei */
        return l;
   else
    {
     l2->vircr = l;
     return p;
    }
 } /* intr2 */


/*-------------------------------------------------------------*/
/*                                                             */
/*    intr3 introduce nodul indicat de l în lista ordonata */
/*    descrescator functie de varsta, si indicata de p     */
/*                                                             */
/*-------------------------------------------------------------*/

 nod *intr3( nod *p,nod *l)
  /* p este inceputul listei in care se face introducerea */
  /* l este nodul care este inserat */
 {
   nod *l1, *l2;

   for(l1 = l2 = p; l1 != NULL && l1->virsta> l->virsta ;
     l2 = l1, l1 = l1->virdes) ;
   l->virdes = l1;
   if(l1 == l2 )
     /* daca nodul este inserat la inceputul listei */
```

```
      return l;
   else
    {
      l2->virdes = l;
      return p;
    }
 } /* intr3 */


/*-------------------------------------------------------------*/
/*                                                             */
/* introdu creaza un nod nou pentru o persoana noua si il      */
/* inlantuie în cele trei evidente                             */
/*                                                             */
/*-------------------------------------------------------------*/

 void introdu(char *n, char *lm, int v)
  /* n reprezinta numele persoanei */
  /* lm reprezinta locul de munca */
  /* v reprezinta varsta */
 {
    nod *t;
    if((t = cauta( n )) != NULL)
     /* daca exista deja o persoana cu numele n */
    {
       /* actualizeaza locul de munca */
       free(t->loc_munca);
       if((t->loc_munca=(char*)malloc(strlen(lm)+1))== NULL)
        {
         printf("Eroare: memorie insuficienta\n");
         exit( 1 );
        }
       else
        {
        strcpy(t->loc_munca, lm); /* noul loc de munca */
        t->virsta = v; /* noua varsta */
        inc2 = scot2(inc2, t->nume); /* elimina nodul din
                 lista ordonata crescator dupa varsta */
        inc2 = intr2(inc2, t); /* introdu nodul in lista
                  ordonata crescator dupa varsta */
        inc3 = scot3(inc3, t->nume); /* elimina nodul din
                  lista ordonata descrescator dupa varsta */
        inc3 = intr3(inc3, t); /* introdu nodul in lista
                 ordonata crescator dupa varsta */
        }
     }
     else /* daca nodul nu apare în evidenta */
       /* aloca memorie pentru nod */
      if((t = (nod*)malloc(sizeof(nod))) == NULL ||
      (t->nume = (char*)malloc(strlen(n) + 1)) == NULL ||
      (t->loc_munca = (char*) malloc(strlen(lm) + 1)) ==NULL)
       {
        printf("Eroare: memorie insuficienta\n");
        exit( 1 );
        }
      else
      {
        /* initializeaza campurile nodului nou */
       strcpy(t->nume, n);
```

```
       strcpy(t->loc_munca, lm);
       t->virsta = v;
       inc1=intr1(inc1, t);/*nodul este introdus în lista 1*/
       inc2=intr2(inc2, t);/*nodul este introdus în lista 2*/
       inc3=intr3(inc3, t);/*nodul este introdus în lista 3*/
     }
  } /* introdu */


/*-------------------------------------------------------------*/
/*                                                             */
/*  parcurge lista ordonata alfabetic si afiseaza evidenta */
/*                                                             */
/*-------------------------------------------------------------*/



 void afis_alf(void)
 {
   nod *p;

   for(p = inc1; p != NULL ; p = p->alf)
   /* parcurge lista ordonata alfabetic */
   {
       printf("Nume: %s\n", p->nume);
       printf("Loc de munca: %s\n", p->loc_munca);
       printf("Virsta: %d\n\n", p->virsta);
   }
 } /* afis_alf */


/*-------------------------------------------------------------*/
/*                                                             */
/*  afiseaza evidenta în ordinea descrescatoare a virstei  */
/*                                                             */
/*-------------------------------------------------------------*/

 void afis_des(void)
 {
   nod *p;

  for(p = inc3; p != NULL; p = p->virdes)
   /*parcurge  lista ordonata descrescator în
       functie de varsta*/
    printf("Nume: %s\n", p->nume);

 } /* afis_des */


/*-------------------------------------------------------------*/
/*                                                             */
/*     afiseaza ordonat alfabetic persoanele cu locul      */
/*                      de munca lm                            */
/*                                                             */
/*-------------------------------------------------------------*/

void afis_loc_munca(char *lm)
{
 nod *l;
```

```
 for(l = inc1; l != NULL; l = l->alf)
   /* parcurge lista ordonata alfabetic */
   if(strcmp(l->loc_munca, lm) == 0) /* daca are locul de
         munca egal cu lm */
      printf(" Nume: %s\n",l->nume);

} /* afis_loc_munca */



/*-------------------------------------------------------------*/
/*                                                             */
/*  afis_virsta parcurge lista ordonata crescator functie      */
/*  de virsta si afiseaza toate persoanele care au             */
/*  virsta >= ca si v                                          */
/*                                                             */
/*-------------------------------------------------------------*/

void afis_virsta(int v)
{
 nod *l;

 for(l = inc2; l != NULL; l = l->vircr)
   /* se parcurge lista ordonata functie de virsta */
   if(l->virsta >= v)
     /* daca virsta este mai >= ca si limita v */
      printf("Nume: %s\n", l->nume);
} /* afis_virsta */


/*-------------------------------------------------------------*/
/*                                                             */
/* elimin scoate persoana cu numele egal cu s din evidenta     */
/*                                                             */
/*-------------------------------------------------------------*/

void elimin(char *s)
{
 if (cauta(s)) /* daca s apare în evidenta */
 {
   inc1 = scot1(inc1, s); /* scoate nodul din lista 1 */
   inc2 = scot2(inc2, s); /* scoate nodul din lista 2 */
   inc3 = scot3(inc3, s); /* scoate nodul din lista 3 */
   /* elibereaza memoria ocupata de nodul eliminat */
   free (auxp->nume);
   free (auxp->loc_munca);
   free (auxp);
 }
 else
    printf("Persoana %s nu apare în evidenta\n");
} /* elimin */


/*-------------------------------------------------------------*/
/*                                                             */
/*  elimin_loc_munca elimina persoanele avind locul            */
/*            de munca egal cu s                               */
```

```
/*                                                                  */
/*------------------------------------------------------------------*/

void elimin_loc_munca(char *s)
{
 nod *l, *aux;

 for(l = inc1; l != NULL; )
  if(strcmp(l->loc_munca, s) == 0) /* s-a gasit o persoana cu
      locul de munca cautat */
   {
      aux = l->alf;
      elimin(l->nume); /* elimina persoana gasita */
      l = aux;
   }
  else
      l = l->alf;
} /* elimin_loc_munca */
```

Fişierul *lab_9.c*

```
#include <stdio.h>
#include <conio.h>
#include "tip.h"
/*------------------------------------------------------------------*/
/*                                                                  */
/*   Functia meniu afiseaza meniul programului, citeste    */
/*   comanda si apeleaza rutina care implementeaza         */
/*   functionalitatea comenzii                             */
/*                                                                  */
/*------------------------------------------------------------------*/

void meniu( void )
{

  char c, s[ 30 ], lm[ 30 ];
  int v;
  nod *t;

  while( 1 )
  {
   clrscr();
   printf("a- Introducerea unei persoane noi în evidenta\n");
   printf("t- Cautarea unui nume în evidenta\n");
   printf("l- Tipareste persoanele ce au un anumit loc de
   munca\n");
   printf("p- Afiseaza persoanele de virsta mai mare sau
   egala cu o valoare citita\n");
   printf("s- Elimina o persoana din evidenta\n");
   printf("d- Elimina persoanele cu un anumit loc de
   munca\n");
   printf("n- Afiseaza alfabetic persoanele din
   evidenta\n");
   printf("v- Afiseaza persoanele în ordinea
   descrescatoare a virstei\n");
   printf("f- Termina programul\n\n\n");
   printf(" Introduceti optiunea:");
```

```
  c =getchar(); getchar(); /* citeste optiunea */

  switch( c )
   {
    case 'a': printf(" Numele persoanei: "); gets(s);
              /* citeste numele */
              printf(" Locul de munca: "); gets(lm);
              /* citeste locul de munca */
             printf(" Virsta:"); scanf("%d",&v); getchar();
              /* citeste virsta */
             introdu(s, lm, v); /* persoana si datele
               aferente sunt introduse în evidenta */
              break;
    case 'v': afis_des(); /* tipareste evidenta în ordinea
                descrescatoare a virstei */
              break;
    case 't': printf(" Numele persoanei: "); gets(s);
               /* citeste numele */
              if ((t = cauta(s)) != NULL)
                 /* daca persoana apare în evidenta */
               {
                printf("Nume: %s\n", t->nume);
                printf("Loc de munca: %s\n", t->loc_munca);
                printf("Virsta: %d\n", t->virsta);
               }
              else
                printf("%s nu apare în evidenta\n",s);
                break;
   case 'l':  printf(" Locul de munca: "); gets(lm);
               /* citeste numele */
              afis_loc_munca(lm); /* afiseaza persoanele cu
                locul de munca citit */
              break;
   case 'p':  printf(" Virsta limita: "); scanf("%d", &v);
               /* citeste virsta */
              getchar();
              afis_virsta( v ); /* afiseaza persoanele cu
                 virsta >= ca limita */
              break;
   case 's':  printf(" Numele persoanei: "); gets(s);
               /* citeste numele */
              elimin(s); /* elimina persoana cu numele s din
                evidenta */
              break;
   case 'd':  printf(" Locul de munca: "); gets(lm);
               /* citeste numele */
               elimin_loc_munca(lm); /* elimina persoanele
                cu un anumit loc de munca */
              break;
   case 'n':  afis_alf(); /* afiseaza evidenta în ordine
                alfabetica */
              break;



    case 'f':  return; /* incheie meniu */
  }
getchar();
```

```
 }
} /* meniu */


/*-------------------------------------------------------------*/
/*                                                             */
/*    Programul principal apeleaza functia meniu               */
/*                                                             */
/*-------------------------------------------------------------*/

void main()
{
   meniu();
} /* main */
```

Comentarea programului

Problema cere tipărirea persoanelor din evidenţă, ordonate după trei criterii: ordonate după nume (comanda *n)*, ordonate descrescător după vârstă (comanda *v*) şi ordonate crescător după vârstă (comanda *p*). Din acest motiv, implementăm evidenţa printr-o listă multiplu înlănţuită (fiecare nod fiind ataşat unei persoane), iar criteriile de înlănţuire a nodurilor corespunzând celor trei moduri de afişare cerute.

Programul este împărţit pe trei fişiere. *tip.h* defineşte tipul nodurilor din listă şi declară operatorii pentru prelucrarea tipului abstract, *lista.c* defineşte tipul abstract listă multiplu înlănţuită, iar *lab_9.c* defineşte rutina pentru interfaţa cu utilizatorul şi programul principal.

Nodurile listei cuprind următoarele câmpuri:

nume - pentru numele angajatului
loc_munca - pentru locul de muncă
virsta - pentru vârsta angajatului
alf - pentru înlănţuirea în ordine alfabetică
vircr - pentru înlănţuirea în ordine crescătoare după vârstă
virdes - pentru înlănţuirea în ordine descrescătoare după vârstă.

Datorită înlănţuirilor multiple, fiecare nod apare înlănţuit în listele ataşate celor trei criterii de ordonare. Variabilele *inc1*, *inc2* şi *inc3* (din fişierul *lista.c*) indică spre începuturile listelor ordonate alfabetic, crescător după vârstă şi, respectiv, descrescător după vârstă. *cauta* parcurge lista ordonată alfabetic (folosind câmpul *alf*) şi caută persoana cu numele egal cu parametrul *n*. Dacă un astfel de nod există, atunci *cauta* returnează pointerul la el, altfel **NULL.**

Când introducem un nod nou, el trebuie înlănţuit după fiecare din cele trei criterii. *intr1* introduce nodul indicat de parametrul *l* în lista indicată de parametrul *p*, astfel încât lista să rămână ordonată alfabetic. Funcţia returnează pointerul către începutul listei rezultate. În mod asemenător, *intr2* şi *intr3* înlănţuie nodul indicat de *l* după criteriile ordonat crescător şi descrescător după vârstă. Operatorul pentru introducerea unui nod nou în lista multiplu înlănţuită se numeşte *introdu*. El are trei parametri, *char *n* pentru numele, *char *lm* pentru locul de muncă şi *int v* pentru vârsta nodului nou. *introdu* începe prin a verifica dacă în evidenţă apare deja o persoana cu numele *n*. În caz afirmativ se actualizează câmpurile *loc_munca* şi *virsta* cu noile valori. În urma acestor modificări, poziţiile nodului în listele ordonate crescător şi descrescător după vârstă se schimbă. De aceea, scoatem nodul din poziţiile anterioare şi îl reintroducem conform noilor câmpuri. Dacă persoana nu apare în evidenţă, atunci se crează un nod nou, se inţializează câmpurile lui, apoi nodul este înlănţuit după cele trei criterii de ordonare.

În mod asemănător, dacă un nod trebuie eliminat din evidenţă, el este scos din toate cele trei înlănţuiri. *scot1* elimină nodul din înlănţuirea *alf* (cea ordonată alfabetic dupa nume), *scot2* scoate nodul din înlănţuirea după *vircr* (ordonată crescător după vârstă) şi *scot3* elimină nodul din înlănţuirea după *virdes* (ordonată descrescător după vârstă). Operatorul (*elimin*) pentru ştergerea persoanei cu numele dat de parametrul *char *s* apelează *scot1*, *scot2* şi *scot3*, după care eliberează memoria ocupată de nodul şters. Pentru aceasta, foloseşte variabila globală *auxp* pe care o poziţionează *scot3*.

afis_alf este operatorul pentru afişarea persoanelor ordonate după nume. *afis_des* tipăreşte persoanele ordonate descrescător după vârstă. *afis_virsta* parcurge lista ordonată crescător după vârstă şi afişează persoanele care au câmpul *virsta* mai mare ca şi parametrul *int v*. *afis_loc_munca* tipăreşte ordonat alfabetic persoanele cu locul de muncă dat de parametrul *char *lm*.

elimin_loc_munca scoate din evidenţă toate persoanele cu locul de muncă egal cu parametrul *char *s*. Rutina parcurge lista ordonată alfabetic şi elimină persoanele apelând *elimin*.

Rutina *meniu* afişează meniul programului şi funcţie de comanda selectată se apelează operatorul tipului abstract ce implementează funcţionalitatea dorită.

B. Probleme propuse

1. Să se realizeze următorul program:

Se citeşte un text care se încheie cu **EOF**. În continuare, programul răspunde unui număr de comenzi după cum urmează:

't' - Tipăreşte identificatorii din evidenţă şi pentru fiecare identificator afişează numărul de apariţii. Tipărirea identificatorilor este în ordine alfabetică.

'l' - Tipăreşte identificatorii din evidenţă şi pentru fiecare identificator afişează numărul de apariţii. Tipărirea identificatorilor este în ordinea lungimii.

's' - Citeşte o valoare întreagă *n*. Elimină din evidenţă toţi identificatorii care conţin *n* vocale.

'a' - Tipăreşte identificatorii din evidenţă în ordinea descrescătoare a numărului de vocale.

'v' - Citeşte o vocală *c* şi un număr întreg *n*. Se tipăresc în ordinea lungimii acei identificatori care conţin vocala *c* cel puţin de *n* ori.

2. Clasamentul jucătorilor de tenis este alcătuit pe baza rezultatelor din turnee. După fiecare turneu disputat, jucătorii sunt răsplătiţi cu un număr de puncte, funcţie de clasificarea lor. De exemplu, locul I ia 10 puncte, locul II 8 puncte , etc. În orice moment clasamentul ia în evidenţă doar primii *k* jucători (*k* este mai mic ca numărul tuturor jucătorilor). Dacă un jucător iese din clasament, el pierde numărul de puncte adunate. Să se realizeze un program care efectuează următoarele operaţii: citeşte rezultatele din turneele disputate, actualizează clasamentul după fiecare turneu, elimină după fiecare turneu jucătorii care sunt situaţi mai jos de locul *k* şi afişează situaţia actualizată.

A. Problemă

Să se realizeze un program care răspunde la următoarele comenzi:

'a' - Se citeşte o linie de forma:

identificator număr

unde *număr* este un număr intreg. Ca rezultat, se reţine în evidenţă identificatorul împreună cu numărul asociat lui.

't' - Se citeşte o linie ce conţine un identificator. Dacă acesta apare în evidentă, se tipăreşte valoarea asociată lui; în caz contrar se tipăreşte un mesaj de eroare.

's' - Se citeşte o linie ce conţine un identificator şi îl şterge din evidenţă.

'l' - Se tipăresc identificatorii din evidenţă în ordine alfabetică împreună cu valorile asociate lor.

'+' - Se citesc două linii, fiecare conţinând un identificator. În cazul în care ambii se află în evidenţă se tipăreşte suma lor. În caz că unul sau ambii identificatori lipsesc din evidenţă, se afişează un mesaj de eroare.

'-' - Se citesc două linii, fiecare conţinând un identificator. În cazul în care ambii se află în evidenţă se tipăreşte diferenţa lor. În caz că unul sau ambii identificatori lipsesc din evidenţă, se afiseaza un mesaj de eroare.

''* - Se citesc două linii, fiecare conţinând un identificator. În cazul în care ambii se află în evidenţă se tipăreşte produsul lor. În caz că unul sau ambii identificatori lipsesc din evidenţă se afişează un mesaj de eroare.

'/' - Se citesc două linii, fiecare conţinând un identificator. În cazul în care ambii se află în evidenţă se tipăreşte rezultatul împărţirii lor. În caz că unul sau ambii identificatori lipsesc din evidenţă se afişează un mesaj de eroare.

'f' - Se termină programul.

Observaţii:

Identificatorii sunt păstraţi în tabelă în mod ordonat.

Se vor prezenta şi comenta structurile de date folosite.

Se va prezenta şi comenta structura de ansamblu a programului, fişierele componente, funcţiile ce compun programul şi efectul lor.

Sursa programului

1. Fişierul *arbore.h*

```
#define Max 20 /* lungimea maxima a unui identificator */

typedef struct pn
{
   char *id; /* cimpul pemtru identificator */
   int valoare; /* cimpul pentru valoare */
   struct pn *sting, *drept; /* fiul sting si drept al
                              nodului */
}nod; /* tipul unui nod din arborele binar ordonat */

/* functiile prin care tipul abstract arbore este prelucrat*/

void sterge(char*);
void listare(void);
```

```
nod *cauta(char*);
void introdu(char*, int);
```

2. Fisierul *arbore.c*

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <ctype.h>
#include <conio.h>
#include "arbore.h"


static nod *root = NULL; /* radacina arborelui */


/*-------------------------------------------------------------*/
/*                                                             */
/*  prezent verifica daca în arborele cu radacina indicata */
/* de t exista un nod avind cimpul id egal cu parametrul   */
/* id                                                          */
/*                                                              */
/*-------------------------------------------------------------*/

nod *prezent (nod *t, char *id)
{
  if (t == NULL) /* nodul nu a fost gasit */
      return 0;
  else
     if (strcmp (t->id, id) < 0)
        /* incearca pentru subarborele drept */
        return prezent (t->drept, id);
     else
       if (strcmp (t->id, id) > 0)
          /* incearca pentru subarborele sting */
          return prezent (t->sting, id);
       else
          /* nodul a fost gasit */
          return t;
} /* prezent */


/*-------------------------------------------------------------*/
/*                                                             */
/*   adauga introduce în arborele indicat de t un nod nou  */
/*   avind cimpul id si valoarea v                          */
/*                                                             */
/*-------------------------------------------------------------*/

nod *adauga(nod *t, char *id, int v)
{

     if (t == NULL) /* nodul nu exista si este creat */
      if ((t = (nod*)malloc(sizeof(nod))) == NULL ||
      (t->id = (char*)malloc(strlen(id)+1)) == NULL)
      {
        printf("Eroare: memorie insuficienta\n");
        exit(1);
```

```
     }
     else /* s-a putut crea nodul */
     {
       strcpy (t->id, id);
       t->valoare = v;
       t->sting = t->drept = NULL;
     }
     else
      if (strcmp (t->id, id) < 0)
          /* inserarea e în subarborele drept */
          t->drept = adauga (t->drept, id, v);
      else
        if (strcmp (t->id, id) > 0)
            /* inserarea e în subarborele sting */
            t->sting = adauga (t->sting, id, v);
        else
         printf("Identificatorul %s apare în evidenta\n",id);
    return t;
} /* adauga */




/*------------------------------------------------------------*/
/*                                                            */
/*   supred rutina auxiliara pentru eliminarea nodurilor   */
/*   cu doi fii                                               */
/*                                                            */
/*------------------------------------------------------------*/

nod *supred (nod *t, nod *p)
{
  nod *q, *q_aux;

  q = t;
  if (q->drept != NULL)
      q->drept = supred (q->drept, p); /* gaseste nodul cel
                  mai mare din subarborele indicat de t */
  else      /* pentru nodul cel mai mare din subarborele
           indicat de t */
  {
    free (p->id);
    /* muta cimpurile nodului indicat de q în nodul indicat
     de p */
    p->id = q->id;
    p->valoare = q->valoare;
    q_aux = q;
    q = q->sting; /* fiul sting este returnat */
    free (q_aux); /* elibereaza memoria ocupata de nod */
   }
  return q;
} /* supred */


/*------------------------------------------------------------*/
/*                                                            */
/* elimina scoate nodul avind cimpul id egal cu parametrul */
/* id din arborele indicat de parametrul p                 */
```

```
/*                                                                  */
/*------------------------------------------------------------------*/

nod *elimina( nod *p , char *id)
{
  nod *q, *q1;
  char *s;

  if (p == NULL)
    printf("Eroare: %s nu apare în evidenta\n", id);
  else
    if (strcmp (p->id, id) < 0)
      /* nodul e în subarborele drept */
       p->drept = elimina (p->drept, id);
    else
      if (strcmp (p->id, id) > 0)
        /* nodul e în subarborele sting */
         p->sting = elimina (p->sting, id);

      else /* s-a gasit nodul ce trebuie sters */
           if (p->sting == NULL) /* daca nu are fiu sting */
             {
               q = p->drept;
               free (p->id);
               free (p);
               return q;
             }
           else
             if (p->drept == NULL) /* nu are fiu drept */
               {
                 q = p->sting;
                 free (p->id);
                 free (p);
                 return q;
               }
             else /* are ambii fii */
                p->sting = supred (p->sting, p);
  return p;
} /* elimina */



/*------------------------------------------------------------------*/
/*                                                                  */
/* parcurge în inordine arborele cu radacina indicata de t          */
/* si afiseaza nodurile                                             */
/*                                                                  */
/*------------------------------------------------------------------*/

void tipareste (nod *t)
{
 if (t != NULL)
 {
    tipareste (t->sting); /* tipareste subarborele sting */
    printf ("%s : %d\n", t->id, t->valoare);
    tipareste (t->drept); /* tipareste subarborele drept */
 }
} /* tipareste */
```

```
/*------------------------------------------------------------*/
/*                                                            */
/*   Operatorul de listare al tipului abstract                */
/*                                                            */
/*------------------------------------------------------------*/

void listare (void)
{
  tipareste (root);
} /* listare */



/*------------------------------------------------------------*/
/*                                                            */
/*     Operatorul de cautare al tipului abstract              */
/*                                                            */
/*------------------------------------------------------------*/

nod *cauta (char *s)
{
  return prezent (root, s);
} /* cauta */


/*------------------------------------------------------------*/
/*                                                            */
/*    Operatorul de stergere al tipului abstract              */
/*                                                            */
/*------------------------------------------------------------*/

void sterge (char *s)
{
  root = elimina (root, s);
} /* sterge */

/*------------------------------------------------------------*/
/*                                                            */
/*    Operatorul de introducere al tipului abstract           */
/*                                                            */
/*------------------------------------------------------------*/

void introdu (char *s, int v)
{
  root = adauga (root, s, v);
} /* introdu */
```

3. Fişierul *main.c*

```
 # include <stdio.h>
 # include <ctype.h>
 # include <string.h>
 # include <conio.h>
 # include "arbore.h"

/*------------------------------------------------------------*/
/*                                                            */
/*  Functia getlin citeste urmatoarea linie de la tastatura*/
```

```
/*  si recunoaste identificatorul si valoarea asociata lui.*/
/*  Identificatorul este returnat în parametrul s, iar     */
/*  valoarea în parametrul val                             */
/*                                                         */
/*---------------------------------------------------------*/
void getlin(char *s ,int *val)
   {
       int i = 0, j;
       char temp[Max];
     gets(temp);/* citeste urmatoarea linie de la tastatura*/
                /* getlin 1 */
     s[i] = '\0' ; *val = 0; /* initializeaza valorile
                              argumentelor */
     while(temp[i] != '\0') /* atita timp cit nu a fost
                            atins sfirsitul sirului */
                            /* getlin 2 */
       if(isalpha(temp[i]))/* daca incepe un identificator */
                           /* getlin 3 */
           {
             j = 0;
             while(isalnum(temp[i])) s[j++] = temp[i++];
                         /*getlin 4*/
             /* memoreaza identificatorul în s */
             s[j] = '\0'; /* memoreaza sfirsitul de sir */
                         /* getlin 5 */
           }
       else
             if(isdigit(temp[i])) /* daca incepe un numar */
                                  /* getlin 6 */
               while(isdigit(temp[i])) /* getlin 7 */
               {
                  *val = *val * 10 + temp[i] - '0' ;
                    /* calculeaza valoarea numarului */
                    /* getlin 8 */
                 i++;
               }
             else i++; /* altfel se trece peste caracterul
                        curent */ /* getlin 9 */
     } /* getlin */


/*---------------------------------------------------------*/
/*                                                         */
/*Functia comanda_a realizeaza functionalitatea comenzii a */
/*                                                         */
/*---------------------------------------------------------*/

      void comanda_a(void)
      {
       int val;
       char s[Max];

       getlin(s, &val); /* citeste o linie de la tastatura */
       if(strlen(s) != 0) /* daca linia e corecta */
         introdu(s, val);
        else printf(" Eroare : linie incorecta \n");
      } /* comanda_a */
```

```
/*-------------------------------------------------------------*/
/*                                                             */
/*Functia comanda_t realizeaza functionalitatea comenzii t     */
/*                                                             */
/*-------------------------------------------------------------*/

      void comanda_t(void)
      {
        char s[Max];
      nod *p;
        int val;

        getlin(s, &val); /* citeste o linie de la tastatura*/
        if(strlen(s) != 0) /* daca linia e corecta */
         {
          if((p = cauta(s)) != NULL)/*cauta nodul în lista*/
              printf("Identificator: %s Valoare: %d \n",
                p->id, p->valoare);
          else printf("Eroare: Identificator nedefinit \n");
        }
        else printf(" Eroare: linie incorecta \n");
      } /* comanda_t */

/*-------------------------------------------------------------*/
/*                                                             */
/*Functia comanda_s realizeaza functionalitatea comenzii s     */
/*                                                             */
/*-------------------------------------------------------------*/
      void comanda_s(void)
      {
        char s[Max];
        int val;

        getlin(s, &val); /* citeste o linie de la tastatura*/
        if(strlen(s) != 0)  /*daca linia citita e corecta */
           sterge(s); /* sterge nodul din lista */
        else printf(" Eroare: linie incorecta \n");

       } /* comanda_s */


/*-------------------------------------------------------------*/
/*                                                             */
/*   Functia comanda_oper executa operatiile legate de         */
/* comenzile +, -, *  si /. Se citesc cei doi operatori si     */
/* se executa operatia dorita.Rezultatul este afisat           */
/*                                                             */
/*-------------------------------------------------------------*/
       void comanda_oper(char c)
       {
        char s1[ Max ], s2[ Max ];
        int val;
        nod *p1, *p2;

        getlin(s1, &val); /* se citeste primul operand */
```

```
        getlin(s2, &val); /* se citeste al doilea operand */

        if((strlen(s1) != 0) && (strlen(s2) != 0))
          if(((p1 = cauta(s1)) != NULL) &&
           ((p2 = cauta(s2)) != NULL))
          /* se verifica daca operanzii apar în lista */
          {
            switch(c) /* functie de tipul comenzii */
            {
              case '+' : val = p1->valoare + p2->valoare;
                         break;
              case '-' : val = p1->valoare - p2->valoare;
                         break;
              case '*' : val = p1->valoare * p2->valoare;
                         break;
              case '/' : if(p2-> valoare != 0)
                           val = p1->valoare / p2->valoare;
                         else
                           printf("Eroare:Impartire la 0\n");
                         break;
             }
             printf(" Rezultatul operatiei e %d \n", val);
          }
          else printf("Operand nedefinit \n");
        else printf(" Eroare: linie eronata \n");
      } /* comanda_oper */


/*-------------------------------------------------------------*/
/*                                                             */
/*  Functia meniu afiseaza meniul programului, citeste     */
/*  comanda si apeleaza subrutina care implementeaza       */
/*  functionalitatea comenzii                                  */
/*                                                             */
/*-------------------------------------------------------------*/

void meniu(void)
{
 char o;

 while( 1 ) /* meniu 1 */
 {
  clrscr();
  /* se afiseaza meniul programului */
  printf("a ... adauga în evidenta un id si val asociata\n");
  printf("t ... tipareste valoarea asociata unui id \n");
  printf("s ... sterge un id \n");
  printf("l ... listeaza id-rii si valorile asociate lor\n");
  printf("+ ... calculeaza suma pentru 2 id \n");
  printf("- ... calculeaza diferenta pentru 2 id \n");
  printf("* ... calculeaza produsul pentru 2 id \n");
  printf("/ ... calculeaza impartirea pentru 2 id \n");
  printf("f ... termina programul \n");
  printf("\n Introduceti optiunea :");
  o = getchar(); getchar(); /* meniu 2 */
  switch(tolower(o)) /* meniu 3 */
   {
     case 'a': comanda_a(); break;
     case 't': comanda_t(); break;
```

```
     case 's': comanda_s(); break;
     case 'l': listare(); break;
     case '+': case '-': case '*': case '/':comanda_oper(o);
                                            break;
     case 'f' : return;
     default : printf(" Eroare : Comanda inexistenta \n");
   }
  getchar();
 }
}


/*-------------------------------------------------------------*/
/*                                                             */
/*       Programul principal apeleaza functia meniu            */
/*                                                             */
/*-------------------------------------------------------------*/

void main(void)
{
  meniu();
}
```

Comentarea programului

Problema este prezentată şi în lucrarea 8, dar acum ea este rezolvată folosind arbori binari ordonaţi. Programul este compus din trei fişiere:

1) *arbore.h* conţine definiţia de tip pentru un nod al arborelui, precum şi declaraţiile funcţiilor ce aparţin tipului abstract arbore binar ordonat.

2) *arbore.c* conţine definiţia tipului abstract arbore binar ordonat.

Funcţia *prezent* caută nodul din arborele indicat de parametrul *t*, care are câmpul *id* egal cu şirul indicat de parametrul *id*. *prezent* returnează pointerul către nodul în care s-a găsit identificatorul sau **NULL** dacă acest nod nu există.

tipareste parcurge arborele în inordine şi afişează conţinutul lui. *tipareste* afişează întâi subarborele stâng (apelându-se recursiv pentru fiul stâng al nodului curent), apoi nodul curent şi în final subarborele drept (apelându-se recursiv pentru fiul drept al nodului curent).

elimina şterge din arborele cu rădăcina *p* nodul al cărui câmp *id* este egal cu parametrul *id*. *elimina* returnează rădăcina arborelui modificat. Dacă *p* este **NULL** înseamnă că nu există nodul căutat în arbore şi se afişează un mesaj de eroare. Dacă nodul curent are câmpul *id* mai mic decât parametrul *id*, atunci căutarea nodului continuă în subarborele drept. Din acest motiv, *elimina* este apelată recursiv pentru fiul drept al nodului curent. În mod asemănător, dacă câmpul *id* al nodului curent este mai mare ca şi parametrul *id*, atunci *elimina* este apelată pentru fiul stâng. Când nodul ce trebuie eliminat a fost găsit,

distingem trei situaţii. În prima, nodul nu are fiu stâng. Se eliberează memoria ocupată de nod şi returnăm fiul său drept. Situaţia a doua este când nodul nu are fiu drept şi atunci returnăm fiul său stâng. Al treilea caz este când nodul are ambii fii şi atunci căutăm pe cel mai mare nod care este mai mic ca cel care trebuie şters. El va fi mutat în locul celui care trebuie eliminat şi apoi îl ştergem din vechea poziţie. Întotdeauna, acest nod nu are fiu drept (dacă ar avea, acesta ar fi mai mare ca el), iar eliminarea lui se face ca pentru un nod doar cu fiu stâng. Pentru a găsi nodul precizat, vom parcurge în subarborele stâng înlănţuirile pentru fiul drept. Rutina *supred* realizează acest lucru.

adauga introduce un nod nou în arborele indicat de parametrul *t*. Nodul are câmpul *id* egal cu parametrul *id* şi câmpul *valoare* egal cu *v*. Rutina returnează rădăcina arborelui modificat.

Toate aceste funcţii au ca şi parametru formal rădăcina arborelui prelucrat, rădăcină care nu este vizibilă în exteriorul fişierului în care a fost definită (din cauza clasei de memorare **static)**. Folosind aceste funcţii se definesc *cauta*, *sterge*, *listare* şi *introdu*, operatorii prin care tipul abstract arbore ordonat este prelucrat.

3) *main.c* conţine programul principal, funcţia pentru citirea comenzilor, rutina de afişare şi prelucrare a meniului şi rutinele auxiliare pentru implementarea comenzilor. Funcţiile din acest fişier sunt absolut identice cu cele din solutia prezentată în lucrarea 8 (unde pentru evidenţă s-a fost o listă înlănţuită). Acest lucru este consecinţa "ascunderii datelor" prin folosirea tipului abstract arbore binar ordonat.

B. Probleme propuse

1. Să se rezolve problema 1 din setul de probleme propuse la lucrarea 8, folosind arbori binari ordonaţi.

2. O bază de date păstrează informaţii despre mărfurile dintr-un magazin. Baza de date e realizată folosind un arbore binar ordonat. Să se scrie un program care realizează urmatoarele:

- introduce o nouă marfă în baza de date;
- tipăreşte baza de date ordonată după numele mărfii şi separat după preţ;
- şterge toate mărfurile cu termenul de garanţie expirat;
- crează un arbore binar ordonat după preţ, arbore care va conţine doar mărfurile cu un preţ mai mic ca o valoare dată *p*. Valoarea *p* se citeşte.

Lucrarea1 3. Se consideră toate structurile posibile de arbori binari ordonaţi cu nodurile etichetate cu numerele 1,2 ... *n*. Să se determine arborele cu lungimea minimă a drumului intern.

Lucrarea2

A. Problemă

Se citesc de la tastatură expresii formate din numere, paranteze rotunde şi operatorii +, -, * şi /. Semnificaţia lor este cea cunoscută din aritmetică. Să se afişeze forma poloneză (vezi lucrarea 7) a expresiilor citite, după care expresiile sunt evaluate, iar rezultatele lor sunt tipărite.

Sursa programului

```
#include <stdio.h>
#include <ctype.h>
#include <stdlib.h>
#include <conio.h>

#define numar  0   /* constanta pentru un operand */
#define operator 1 /* constanta pentru un operator */


typedef struct n
{
   int val; /* operatorul sau valoarea operandului */
   int fel; /* nod operand sau operator */
   struct n *left, *right; /* fii stang si drept */
}nod; /* tipul pentru un nod din arborele pentru expresii */


int atom, valoare;

nod *expresie(void);
nod *termen(void);
nod *factor(void);


/*------------------------------------------------------------*/
/*                                                            */
/*  mknode creaza un nod nou si ii initializeaza campurile */
/*             cu parametrii transmisi                        */
/*                                                            */
/*------------------------------------------------------------*/

nod *mknode (int val, int fel, nod *stang, nod *drept)
   /* val: valoarea nodului */
   /* fel: felul nodului: numar sau operator */
   /* stang: fiul stang */
   /* drept: fiul drept */
{
 nod *v;
```

```
 if (( v = (nod*)malloc(sizeof(nod))) == NULL)
 {
    printf("Eroare: memorie insuficienta\n");
    exit(1);
 }
 else
 {
     v->val = val;
     v->fel = fel;
     v->left = stang;
     v->right = drept;
     return v; /* returneaza pointerul la nodul nou */
  }

} /* mknode */


/*-------------------------------------------------------------*/
/*                                                             */
/*     evaluare parcurge arborele transmis ca si parametru */
/*          si returneaza valoarea asociata lui            */
/*                                                             */
/*-------------------------------------------------------------*/

int evaluare (nod *a)
{
 int val_stanga, val_dreapta;

 if (a != NULL)
  if (a->fel == operator) /* pentru operator */
  {
     val_stanga = evaluare (a->left); /* evalueaza
                                         subarborele stang */
     val_dreapta = evaluare (a->right); /* evalueaza
                                           subarborele drept */
     switch (a-> val)
     {
      case '+': return val_stanga + val_dreapta;
      case '-': return val_stanga - val_dreapta;
      case '*': return val_stanga * val_dreapta;
      case '/':
              if (val_dreapta != 0)
                return val_stanga / val_dreapta;
              else
              {
               printf("Eroare: impartire la 0\n");
               exit(1);
              }
      default: printf("Eroare: operator necunoscut\n");
                exit(1);
     } /* switch */
  }
  else /* pentru operanzi */
     return a->val; /* returneaza valoarea operandului */

} /* evaluare */


/*-------------------------------------------------------------*/
```

```
/*                                                              */
/*      gettop returneaza urmatorul element (operand,           */
/*      operator sau paranteze) din expresie                    */
/*                                                              */
/*--------------------------------------------------------------*/

int gettop( void )
{
 char c;

 while ((c = getchar() ) == ' ' || c == '\t'); /* trece peste
                                                  spatii si tab */

 if (!isdigit( c )) /* pentru operator sau paranteze */
      return c;
 else
 {
    valoare = c - '0';
    while (isdigit (c = getchar()))
       valoare = valoare * 10 + c - '0'; /* calculeaza
                                        valoarea operandului */
    ungetc( c,stdin );
    return  numar; /* returneaza constanta pentru operand */
  }

} /* gettop */

/*--------------------------------------------------------------*/
/*                                                              */
/*      tipareste parcurge arborele expresiei in ordine         */
/*       postfix, rezultand forma poloneza a expresiei          */
/*                                                              */
/*--------------------------------------------------------------*/

void tipareste( nod *r )
{

   if (r != NULL)
   {
    tipareste (r->left); /* parcurge subarborele stang */
    tipareste (r->right); /* parcurge subarborele drept */
    if (r->fel == operator) /* pentru nod operator */
      printf(" %c  ",r->val);
    else  /* pentru nod operand */
      printf(" %d ",r->val );
   }

} /* tipareste */
/*--------------------------------------------------------------*/
/*                                                              */
/*       expresie trateaza subexpresiile care contin            */
/*                 operatori aditivi                            */
/*                                                              */
/*--------------------------------------------------------------*/

nod *expresie( void )
{
 char op;
 nod *b, *a;
```

```
 a = termen();
 while (atom == '+' || atom == '-')
 {
       op = atom;
       atom = gettop();
       b = termen();
       a = mknode(op, operator, a, b);
 } /* while */

 return a;

} /* expresie */


/*-------------------------------------------------------------*/
/*                                                             */
/*        termen trateaza subexpresiile care contin            */
/*               operatori multiplicativi                      */
/*                                                             */
/*-------------------------------------------------------------*/

nod *termen( void )
{
  char op;
  nod *b, *a;

  a = factor();
  while (atom == '*' || atom == '/')
  {
      op = atom;
         atom = gettop();
      b = factor();
         a = mknode(op, operator, a, b);
   }
   return a;

} /* termen */




/*-------------------------------------------------------------*/
/*                                                             */
/*   factor construieste subarborele pentru un operand         */
/*    sau o expresie intre paranteze                           */
/*                                                             */
/*-------------------------------------------------------------*/
nod *factor( void )
{
 nod *v;

  if(atom == numar) /* pentru operand */
  {
      v = mknode(valoare, numar, NULL, NULL); /* creaza nodul
                                          pentru operand */
      atom = gettop();/*preia urmatorul element din expresie*/
```

```
     return v;
   }
   else
     if(atom == '(') /* pentru expresie intre paranteze */
     {
      atom = gettop(); /* preia urmatorul element din
                          expresie */
      v = expresie(); /* construieste arborele pentru
                          expresie */
      if( atom == ')' ) atom = gettop();/*paranteza inchisa*/
      return v;
     }
    else return NULL;

} /* factor */

/*-------------------------------------------------------------*/
/*                                                             */
/*                     Programul principal                     */
/*                                                             */
/*-------------------------------------------------------------*/

void main( void )
{
 nod *root = NULL; /* radacina arborelui binar pentru o
                      expresie */
 int valoare;


 clrscr();
 printf("Introduceti o expresie:\n\n\n");
 atom = gettop(); /* preia primul element al expresiei */
 while(atom != EOF) /* atata timp cat sunt expresii de
                       prelucrat */
 {
     printf("Expresia in notatie poloneza:\n");
     root = expresie(); /* construieste arborele asociat
                           expresiei */
     tipareste( root ); /* afiseaza forma poloneza a
                           expresiei */
     printf("\n\n");
     valoare = evaluare( root ); /* evalueaza expresia */
     printf("Valoarea expresiei este %d  \n\n", valoare);
     getchar();
     clrscr();
     printf("Introduceti o expresie:\n\n\n");
     atom = gettop(); /* preia primul element al expresiei */
 } /* while */

} /* main */
```

Prezentarea programului

Din punct de vedere teoretic, structura expresiilor este descrisă de următoarele reguli:

```
expresie : expresie + expresie +..... + expresie
         | expresie - expresie - ..... - expresie
         | termen

termen   : termen * termen * ..... * termen
         | termen / termen / ..... / termen
         | factor

factor   : număr
         | ( expresie )
```

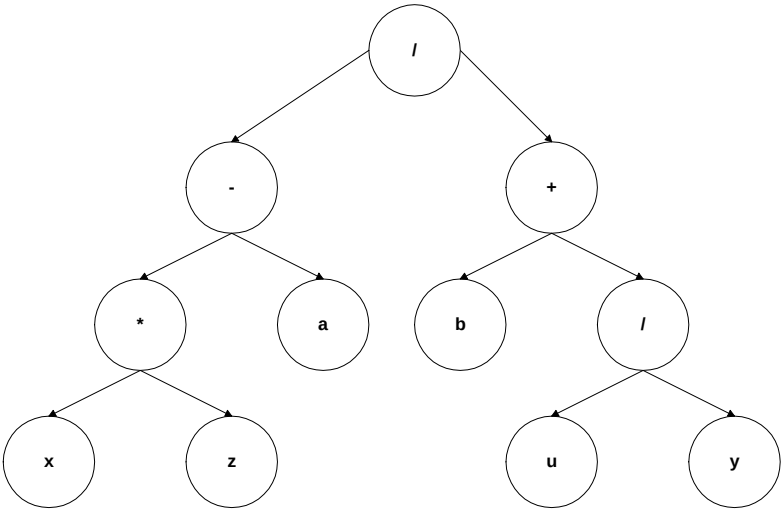
Aceste reguli (numite producţii) sunt stabilite de regulile de precedenţă a operatorilor (care indică ordinea în care sunt executate operaţiile într-o expresie). Astfel, primele două reguli corespund operatorilor aditivi + şi -, care au precedenţă minimă, următoarele două reguli sunt pentru operatorii multiplicativi * şi /, iar ultimele două producţii tratează operanzii şi expresiile între paranteze.

În prima fază, programul construieşte arborele binar ataşat unei expresii. Acesta are ca şi noduri interne nodurile ataşate operatorilor, iar ca noduri frunză cele ataşate operanzilor (numerelor) din expresie.

De exemplu, pentru expresia:

(x * z - a) / (b + u / y)

arborele construit este cel din figura următoare:



Funcţia *factor* tratează un operand (număr) sau o expresie între paranteze. *factor* returnează fie nodul frunză construit pentru operand, fie rădăcina subarborelui asociat expresiei între paranteze.

Funcţia *termen* analizează subexpresiile conţinând operatorii * şi /. Ea returnează rădăcina arborelui construit pentru aceste subexpresiile.

Funcţia *expresie* tratează subexpresiile formate cu operatorii + şi -. e*xpresie* returneaza rădăcina arborelui construit.

După construirea arborelui binar ataşat unei expresii, acesta este parcurs în postordine, astfel afişându-se forma poloneză a expresiei iniţiale.

evaluare are ca şi parametru rădăcina arborelui (sau a subarborelui) pentru care dorim să calculăm valoarea. *evaluare* returnează valoarea calculată. Dacă nodul corespunde unui operator sunt calculate valorile pentru subarborele stâng şi drept. Apoi, aceste valori sunt folosite ca operanzi pentru operatorul indicat de nodul curent, iar rezultatul operaţiei este returnat de *evaluare*. Dacă nodul corespunde unui operator, atunci valoarea lui este returnată.

B. Probleme propuse

1. Se consideră expresii formate din numere, paranteze rotunde şi operatorii +, -, *, / şi ^ (ridicare la putere). Să se transforme expresia furnizată la intrare în una echivalentă, dar fără paranteze redundante.

ex. `((2 + 1) + 5 + ( 8 * 5 ))` trece în `2 + 1 + 5 + 8 * 5`

`(3 + 8 * (5 + 2))` trece în `3 + 8 * (5 + 2)`

1. Să se realizeze un editor de texte. Textul iniţial este încărcat dintr-un fişier, iar textul modificat este salvat într-un fişier nou. Editarea se face prin următoarele comenzi:

. *I, m* - inserează o linie după linia *m*
. *D, m, n* - şterge liniile între liniile *m* şi *n*
. *R, m, n* - înlocuieşte liniile între *m* şi *n* cu linii noi
. *E* - termină procesul de editare

2. Să se genereze toate posibilităţile de planificare a unui turneu la care participă *n* concurenţi, ştiind că doi concurenţi se întâlnesc o singură dată şi un concurent dispută un singur joc pe zi.

3. Pentru elaborarea unui test de aptitudini se dispune de un set de *n* întrebări. Fiecare întrebare este cotată cu un număr de puncte. Să se elaboreze toate chestionarele posibile care au un număr de întrebări cuprins între *a* şi *b* (*a* şi *b* se citesc de la tastatură), astfel încât numărul total de puncte pe chestionar să fie cuprins între *p* şi *q* (*p* şi *q* se citesc de la tastatură).

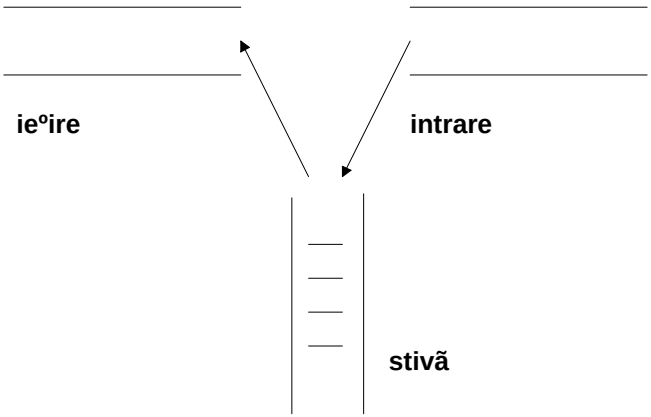
4. Într-un grup de persoane, fiecare persoană cunoaşte eventual alte persoane din grup. Să se formeze toate echipele posibile, astfel încât pentru o echipă, fiecare persoană este cunoscută de cel puţin un membru al acelei echipe.

5. Se dă un grup de oraşe şi conexiunile între ele. Să se găsească toate drumurile care trec prin toate oraşele parcurgându-le o singură dată.

6. Poziţiile oraşelor unei ţări sunt date prin coordonatele lor carteziene. Să se găsească configuraţia reţelei telefonice care îndeplineşte condiţiile:
- orice oraş este conectat la reţeaua telefonică;
- costul reţelei (proporţional cu lungimea liniilor telefonice) este minim.

7. O hartă cuprinde *n* ţări. Să se găsească numărul minim de culori cu care pot fi colorate ţările, astfel încât două ţări vecine să nu fie colorate cu aceeaşi culoare.

8. Se consideră o cale ferată de forma celei din figura următoare. Pe linia de intrare se află *n* vagoane numerotate de la 1 la *n*. Notăm cu *I* operaţia de introducere a vagoanelor de pe prima poziţie pe linia de intrare în stiva, şi cu *S* cea de extragere din vârful stivei către ieşire. Alte operaţii nu sunt permise.



Pentru un anumit număr de vagoane dat, programul va realiza următoarele:

a) Dându-se la intrare o secvenţă de forma :
ISSI...
să se raspundă dacă secvenţa respectivă de operaţii poate fi executată.

b) Dându-se o permutare oarecare a numerelor 1...*n*, să se spună dacă ea poate fi obţinută pe linia de ieşire, pornind de la *n* vagoane aflate în ordine pe linia de intrare. În caz afirmativ, să se dea succesiunea de operaţii *I* şi *S* care duce la configuraţia respectivă.

c) Să se tipărească toate secvenţele de vagoane, care pot fi obţinute pe linia de ieşire, pentru *n* dat iniţial.

9. Problema "Descendenţilor după părinte": Petre ştie că este fiul lui Ion şi mai ştie că este frate cu Virgil şi Matei. Copiii lui Petre sunt Ilie şi Marcu, iar ai lui Virgil sunt: Ionel şi Petrică. Filip ştie că este fiul lui Matei şi frate cu Ioana. Sabin este fiul lui Ionel.
Ion doreşte să ştie numele tuturor nepoţilor săi.
Filip vrea să ştie dacă pe bunicul lui îl cheamă Ilie.
Ionel vrea să ştie dacă este văr cu Alexandru.
Să se scrie programul, care pe baza unor informaţii de acest tip, este în măsură să răspundă la următoarele genuri de întrebări:

1) Care sunt nepoţii bunicului "nume"?
2) "Nume1" este bunicul lui "nume2"?
3) "Nume1" este văr primar cu "nume2"?

Se va verifica dacă datele de intrare sunt consistente. Se consideră că două persoane sunt diferite dacă au nume diferite.

Datele de intrare se citesc dintr-un fişier cu numele REGULI.DAT, având câte o regulă pe fiecare linie. Regulile sunt de forma:

"nume1"=FIU("nume2")
sau
"nume1"=FRATE("nume2")

unde "nume1" şi "nume2" sunt şiruri de maxim 20 de litere (literele mici şi cele mari se consideră identice). Este permisă folosirea unor separatori de forma spaţiu şi tab. Detectarea caracterului $ la începutul unei linii reprezintă sfârşitul şirului de reguli. După ce s-au citit toate regulile, de pe liniile următoare se citesc tipul întrebării (1, 2 sau 3) şi informaţiile aferente acestuia, separate prin virgulă. Fiecare întrebare se citeşte de pe o linie.

Datele de ieşire se vor scrie sub forma:

"nume1"=BUNIC("nume2",...,"numep")

pentru primul tip de intrebare, sau

ESTE, NU ESTE sau *EROARE*

pentru celalalte două tipuri de întrebări. Răspunsurile sunt scrise pe linii distincte.

10. Un dicţionar este scris într-un alfabet pentru care nu se ştie dacă există sau nu o ordine între litere. Să se stabilească pe baza ordinii în care apar cuvintele în dicţionar, dacă există sau nu ordine între literele alfabetului.

11. Un dreptunghi *D*, cu laturile paralele cu axele de coordonate este precizat prin coordonatele (întregi) a două vârfuri opuse. Să se scrie un program care calculează şi afişează aria şi perimetrul conturului reuniunii *D1 U D2 U ...* de dreptunghiuri. Datele de test se citesc dintr-un fişier text, ce conţine pe linii coordonatele vârfurilor unui dreptunghi în ordinea *x1, y1, x2, y2*.

12. Un număr de *2n + 1* persoane participă la discuţii ce durează *K* zile. Să se găsească variantele de aşezare a persoanelor la masa de discuţii, astfel încât o persoană să nu aibă în 2 zile diferite aceiaşi vecini.

13. Să se scrie un program care citeşte de la tastatură în mod repetat expresii cu doi operanzi întregi şi un operator (+, -, *, /). Operanzii sunt exprimaţi în cifre romane, iar numerele negative sunt precedate de caracterul

minus. Programul afişează rezultatele exprimate tot în cifre romane.

14. Se dă o expresie în notaţie poloneză, care conţine operatorii +, -, *, /, <, <=, >, >=, =, <>, AND şi OR. Să se verifice dacă expresia furnizată este corectă:

a) Fiecărui operator să-i corespundă doi operanzi.

b) Să nu existe operanzi cărora să nu le corespundă operatori.

c) Să fie respectate următoarele restricţii legate de tip:

Pentru +, -, *, / ambii operanzi trebuie să fie numere. Rezultatul este tot un număr.

Pentru <, <=, >, >=, =, <> operanzii sunt numere, iar rezultatul este boolean.

Pentru AND şi OR operanzii sunt de tipul boolean, iar rezultatul este tot boolean.

15. Se dă în plan un poligon oarecare *P* cu vârfurile *P1, P2, P3, P4, ... Pn* ale căror coordonate (numere întregi) sunt citite dintr-un fişier. Să se determine dacă un punct dat *A(p,q)* este în interiorul, pe laturile, sau în exteriorul poligonului *P*.

16. Se dă o succesiune de preţuri, fiecare sub forma a două numere întregi, ce reprezintă cantitatea de lei, respectiv de bani. Să se tipărească preţurile în cifre şi cuvinte ca în exemplul de mai jos:

871 25 opt sute saptezeci si unu lei si douazeci si cinci bani
0 0 nimic
0 35 treizeci si cinci bani
113 0 una suta treisprezece lei

17. Studenţii unui an pot alege 10 cursuri facultative: muzică, literatură, pictură, sanscrită, arheologie, canto, poetică, gastronomie, albaneză şi puericultură. Opţiunile fiecărui student se citesc de pe o linie cu următoarea structură:

numele studentului (ca o succesiune de caractere care se încheie cu un punct) şi în continuare, pe zece poziţii, sunt spaţii sau X. Fiecare poziţie se asociază unui curs, în ordinea enumerării de mai sus, iar X pe poziţia respectivă înseamnă că studentul a ales cursul respectiv. Un student poate alege oricâte cursuri. De exemplu:

Dumitrescu Dumitru. X XX

Studentul Dumitrescu Dumitru a ales pictura, canto si poetica.
La începutul dialogului se află numărul studenţilor, care nu depăşeşte 100.

Să se tipărească, pentru fiecare curs, lista studenţilor înscrişi. Cursurile vor apărea în ordinea descrescătoare a numărului de studenţi înscrişi, iar listele de studenţi sunt afişate în ordine alfabetică.

18. Dintr-o bară de lungime *L* se taie *m* repere de lungime *l1, l2, ..., lm*. Să se găsească toate variantele de debitare a barei astfel încât pierderile de material să fie minime. Se impune ca printr-o debitare să se obţină cel puţin câte un exemplar din fiecare reper. Valorile lui *L* şi *l1, l2, ..., lm* sunt întregi şi sunt citite de la tastatură.

19. Un nume de fişier poate conţine şi caracterele ? şi *. Să se scrie un program care citeşte o descriere pentru numele de fişiere şi listează toate fişierele care corespund descrierii. Dacă astfel de fişiere nu există atunci se afişează numele cele mai apropiate de descrierea citită. Apropierea (distanţa) între două nume de fişiere este stabilită de numărul minim de operaţii de ştergere/ inserare/ înlocuire de caractere, necesare pentru a transforma primul cuvânt în al doilea.

Exemplu:
Pentru numele de fişiere:

DIST.BAK
DIST.EXE
DIST.PAS
RANDUL.1
RANDUL.2
RANDUL.3
C
FLORINLL
PAS
PLANIFIC

şi descrierea ***L*I*.*?*** se va afişa (între paranteze este specificată distanţa):

DIST.BAK (1)
DIST.EXE (1)
DIST.PAS (1)
RANDUL.1 (1)
RANDUL.2 (1)
RANDUL.3 (1)
FLORINLL (1)
PLANIFIC (1)

20. Un automat finit constă dintr-o mulţime de stări (care sunt numere întregi în intervalul 1..*n*) şi o matrice de tranziţie. Matricea de tranziţie precizează, pentru fiecare stare şi fiecare caracter de intrare, starea următoare. Presupunem că intrările pot lua două valori 0 sau 1. Anumite stări sunt stări finale. Presupunem că în cazul nostru, stările finale sunt cele al căror număr este divizibil cu 7. Două stări *x* şi *y* sunt echivalente dacă ele sunt identice sau dacă îndeplinesc simultan condiţiile:

(1) Ambele sunt finale sau nefinale.
(2) Pentru o intrare 0, ambele trec în stări echivalente.
(3) Pentru o intrare 1, ambele trec în stări echivalente.

Programul calculează mulţimea stărilor echivalente, pentru un automat finit dat.

1. Să se realizeze un program pentru rezervarea locurilor la o companie de zbor. Programul are următoarele comenzi:

ZBOR < *număr zbor* > < *număr total locuri* >
adaugă în evidenţă cursa < *număr zbor*> în cazul în care aceasta nu exista deja.

ANULARE < *număr zbor* >
anulează cursa. Pasagerii (dacă vor) îşi rezervă locuri la o altă cursă.

LOC < *număr zbor* > < *nume pasager* >
Pasagerul <*nume pasager*> îşi rezervă (dacă mai există loc) un loc pentru cursa < *număr zbor* >. Se permit pasageri cu acelaşi nume.

RENUNT < *număr zbor* > < *nume pasager* >
Pasagerul < *nume pasager* > renunţă la locul său din cursa < *număr zbor* >.

PRINT < *număr zbor* >
Afişează în ordine alfabetică lista pasagerilor cursei < *număr zbor* >.

Dacă s-a introdus o comandă eronată se afişează "eroare" şi se aşteaptă o nouă comandă.

2. Se consideră o structură de date abstractă de tip arbore binar ordonat asupra căreia sunt executate următoarele operaţii: adăugare, căutare şi suprimare. Nodurile sunt ordonate funcţie de valoarea câmpului cheie. Fiecărui nod îi este ataşat un contor de acces la nodul respectiv. La anumite intervale de timp, structura se reorganizează într-o nouă structură de arbore binar ordonat,

în care ordinea de creare a nodurilor este corespunzătoare valorilor descrescătoare ale contoarelor de acces. Se cere:

a. Să se precizeze structurile de date aferente structurii de arbore.

b. Să se definească rutinele care gestionează structura de date arbore (adaugă, caută, suprimă).

c. Să se definească rutina pentru reorganizarea arborelui.

3. Să se realizeze un program pentru corectarea erorilor de ortografie dintr-un text. Cuvintele textului sunt introduse de la tastatură. Fiecare cuvânt este prelucrat astfel:

a. Dacă figurează în dicţionar el este copiat în fişierul de iesire.

b. Dacă nu figurează în dictionar, atunci cuvântul este afişat la terminal şi se cere utilizatorului una din următoarele opţiuni:

- Dacă cuvântul este corect, atunci el trebuie adăugat în dicţionar şi în fişierului de ieşire.
- Dacă cuvântul este incorect, atunci se introduce o nouă formă corectă a cuvântului care se adaugă la fişierul de ieşire.
- Dacă cuvântul este incorect, atunci se introduce o nouă formă corectă a cuvântului care se adaugă la fişierul de ieşire şi în plus forma incorectă este reţinută, urmând ca de ori câte ori este întâlnită, să fie înlocuită automat cu forma corectă.

4. Un polinom *P(x)* se reprezintă ca o succesiune de tupleţi (coeficient, putere), ambii întregi. Să se realizeze un program care execută înmulţiri şi adunări între polinoame. Polinoamele se introduc sub forma unei succesiuni de numere întregi. Primul indică numărul termenilor, iar următoarele *N* perechi de numere reprezintă fiecare câte un tuplet. Un polinom este urmat de '+' sau '*' , funcţie de operaţia dorită. După calcularea şi afişarea rezultatului se citeşte o valoare reală *x*, pentru care se evaluează derivata polinomului.

Observaţie: În intrare, polinoamele se dau cu termenii în ordinea oarecare a puterii şi pot apărea mai mulţi termeni pentru aceeaşi putere a lui *x*.

5. Să se realizeze un program care răspunde la următoarele comenzi:

i - Citeşte o linie cu următoarea structură:

identificator număr

număr este de forma: *secvenţă de cifre* sau *secvenţă de cifre . secvenţă de cifre*

Ca răspuns la comandă, identificatorul este introdus în tabelă împreună cu valoarea asociată lui. Dacă identificatorul apare în evidenţă, se tipăreşte un mesaj de eroare şi comanda este încheiată.

t - Tipăreşte în ordine alfabetică identificatorii din evidenţă.

s - Preia din intrare un identificator. Dacă el există în evidenţă el este şters, în caz contrar se tipăreşte un mesaj de eroare.

m - Preia din intrare o linie de forma:
identificator număr
Dacă identificatorul apare în evidenţă, valoarea lui este actualizată la noua valoare, în caz contrar, el este luat în evidenţă.

v - Tipăreşte identificatorii în ordinea inversă a valorii asociate lor.

e - Citeşte din intrare o expresie în notatie postfix. Operanzii pot fi identificatori sau numere având forma precizată, iar operatorii sunt + - * /. Valoarea asociată identificatorilor este extrasă din evidenţă. Dacă în expresie apare un identificator neluat în evidenţă, se tipăreşte un mesaj de eroare şi se abandonează evaluarea expresiei. Se va calcula expresia, iar valoarea obţinută se afişează.

f - Termină programul.

6. Să se realizeze un program pentru o tabelă de referinţe încrucişate după cum urmează: citeşte un text oarecare, ce se încheie cu **EOF**. După recepţionarea sfârşitului de text se tipăreşte, în ordine alfabetică, lista identificatorilor întâlniţi şi, pentru fiecare identificator, numerele liniilor în care apare.
Observaţie: Tabela identificatorilor este organizată sub forma unei liste. După fiecare 5 operaţii în listă, aceasta se va reorganiza în ordinea descrescătoare a numărului de referiri ale nodurilor.

7. Să se realizeze următorul program:
Se citeşte un text oarecare, care se încheie cu **EOF**. În continuare, programul răspunde la un număr de comenzi după cum urmează:

t - Tipăreşte în ordine alfabetică identificatorii din evidenţă şi pentru fiecare numărul de apariţii.

l - Tipăreşte, în ordinea lungimii identificatorilor, identificatorii din evidenţă şi pentru fiecare numărul de apariţii.

s - Citeşte o valoare *n* întreagă. Elimină din evidenţă toti identificatorii care conţin *n* vocale.

a - Tipăreşte identificatorii din evidenţă în ordinea descrescătoare a numărului de vocale conţinut.

v - Se citeşte o vocală *c* şi un număr întreg *n*. Se cere să se tipărească, în ordinea lungimii identificatorilor, acei identificatori care conţin vocala *c* cel puţin de *n* ori.

8. Să se scrie un program care realizează următoarele:

Citeşte un prim text care se încheie cu **EOF**. Memorează identificatorii prezenţi în acel text. În cazul în care un identificator apare de mai multe ori în acest prim text, el se reţine o singură dată. În continuare, se citeşte un al doilea text care se încheie cu **EOF**. Se cere să se tipărească lista identificatorilor prezenţi în primul text şi pentru fiecare identificator să se menţioneze liniile din cel de-al doilea text în care este prezent. Lista este tipărită în ordine alfabetică.

9. Se citeşte un text format din linii de forma:

nume; loc_de muncă; adresa;

Textul se încheie cu **EOF**. După citirea textului se cere să se tipărească în ordine alfabetică:

a. Lista nominală de forma:

nume adresa loc_de_muncă

b. Lista locurilor de muncă în forma:

loc_de_munca număr_angajaţi

10. Se citeşte o succesiune de linii care se încheie cu **EOF**. Aceste linii au ca prim caracter fie #, fie * şi în rest conţin un text oarecare. Se vor construi trei evidenţe separate conţinând: identificatorii prezenţi exclusiv în linii care încep cu #, cei prezenţi exclusiv în linii care încep cu *, şi cei prezenţi în linii ce încep cu #, cât şi în linii ce încep cu *. Fiecare identificator apare o singură dată în evidenţă. În continuare se citeşte un text oarecare, format din mai multe linii, care se încheie de asemenea cu **EOF**. Se cere să se tipărească separat, în ordine alfabetică, cele trei evidenţe, şi, pentru fiecare identificator, să se specifice de câte ori apare în textul citit. Apoi se citeşte un al doilea text care se încheie cu **EOF**. Se cere să se tipărească o listă unică cuprinzând toţi identificatorii din cele trei liste, şi, pentru fiecare identificator, să se specifice de câte ori apare în ultimul text citit. Lista se va tipări în ordinea numărului de apariţii.

11. Se citesc dintr-un fişier datele corespunzătoare rezultatelor obţinute de candidaţii la examenul de admitere. Datele se introduc ca o succesiune de linii de forma:

nume; sectie; medie

Succesiunea de linii se încheie cu **EOF**.

Secţia este: "calculatoare" sau "automatică", iar media este un număr real. Dacă după citirea liniei se constată că secţia nu s-a dat corect, se cere rectificarea până când operatorul răspunde cu una din cele două variante acceptate. Dacă după citirea liniei se constată că numele este deja prezent în

evidenţă, se actualizează vechea înregistrare conform noilor informaţii. După încheierea citirii se vor tipări:

a. Lista candidaţilor admişi, în ordinea mediilor în forma:
nume sectia medie

Se ştie că numărul de locuri este 40 la calculatoare şi 40 la automatică. Sunt admişi primii 80 de candidaţi în ordinea mediilor, indiferent de sectia pentru care au optat. Repartizarea pe secţii se face în limita locurilor, în funcţie de opţiunea candidatului, cu prioritate pentru cei cu medie mai mare.

b. Lista candidaţilor respinşi în ordine alfabetică, sub forma:
nume medie

12. Se citeşte o secvenţă de linii de forma:
obiect atribut1 atribut2 ....
prin care se asociază unui obiect o secvenţă de atribute. Acelaşi obiect poate apărea pe mai multe linii. Acelaşi atribut poate apărea de mai multe ori pe liniile ce caracterizează acelaşi obiect. Atributul multiplu este reţinut o singură dată. Secvenţa de linii se încheie cu **EOF**. Se va scrie un program care răspunde interactiv la următoarele comenzi:

a - Citeşte un atribut şi afişează în ordine alfabetică toate obiectele care au acel atribut.

s - Citeşte un atribut şi îl elimină pentru toate obiectele din evidenţă. Dacă un obiect nu mai are nici un atribut el este eliminat din evidenţă.

l - Citeşte o linie care conţine doar atribute. Se afişează toate obiectele care conţin atributele de pe linie. Listarea obiectelor se face astfel: întâi obiectele care conţin doar atributele din secvenţă, apoi pe cele care au un atribut suplimentar, apoi cele care au două atribute suplimentare, etc.

b - Afişează în ordine alfabetică toate atributele şi, pentru fiecare, numărul obiectelor care conţin acel atribut.

p - Citeşte o linie ce conţine doar atribute. Se elimină din evidenţă obiectele ce conţin atributele specificate.

e - Termină programul.

13. Se citeşte un fişier care conţine linii de forma:
nume prenume materia1 nota1 materia2 nota2 .....
Secvenţa de linii se încheie cu o linie vidă. Fiecare linie conţine numele, prenumele, materiile şi notele obţinute de studenţii unui an de studiu. Dacă un student nu s-a prezentat la un examen, linia corespunzătoare va conţine doar numele materiei fără a apărea şi nota.

De exemplu:
Mihailescu Ion proiectare 5 compilatoare algoritmi 6

În continuare, programul răspunde interactiv la următoarele comenzi:

afişează_top - Afişează lista ordonată funcţie de medie, a studentilor care au promovat toate examenele. Studenţii restanţieri sunt afişaţi într-o listă separată, ordonată funcţie de nume, pentru fiecare student indicându-se numărul restantelor.

restanţe - Calculatorul afişează prompterul > aşteptând introducerea unui nume de materie. După ce s-a citit numele materiei sunt afişaţi, ordonat alfabetic, toţi studenţii care au acea materie restanţă. În continuare, este afişat prompterul > şi se aşteaptă introducerea unei noi materii. Comanda se încheie atunci când se tastează o linie vidă.

modifică - Citeşte numele şi prenumele unui student. Apoi calculatorul afişează prompterul > şi aşteaptă introducerea unei materii şi a notei. Pentru materia specificată se modifică nota studentului. Prompterul > este afişat până la introducerea unei linii vide.

elimină - Citeşte un număr. Din evidenţă sunt eliminaţi toţi studenţii cu un număr de restanţe mai mare sau egal cu numărul din comandă.

terminare - Termină programul.

14. Se citeşte o secvenţă de linii de forma:

id = cuvânt1, cuvânt2, ...

prin care unui cuvânt i se asociază o listă de sinonime. Există posibilitatea ca un identificator să apară de mai multe ori în partea stângă a semnului =, caz în care noua listă a sinonimelor se adaugă la cea veche. Secvenţa de linii se încheie cu **EOF**. În continuare, programul răspunde repetat la următoarele comenzi:

c - Citeşte un cuvânt şi afişează ordonat alfabetic lista sinonimelor ataşate lui.

v - Afişează ordonat funcţie de numărul de consoane toţi identificatorii din evidenţă (cei care apar în stânga semnului =).

s - Citeşte un cuvânt şi elimină din evidenţă acel cuvânt împreună cu lista ataşată de sinonime.

x - Se citeşte un cuvânt şi se cere afişarea tuturor cuvintelor cu care el este sinonim.

e - Termină programul.

15. Se citesc linii cu următoarele forme:

(1) număr: expresie în forma poloneză
(2) delete: număr
(3) afiş: număr
(4) evaluează: număr
(5) identificator: număr

Pentru linia cu forma (1) se construieşte o listă înlănţuită, cu un nod pentru fiecare element (operand sau operator) astfel: pentru un operator se construieşte un nod nou, iar pentru un operand se verifică dacă a apărut într-o expresie anterioară. În caz afirmativ, se consideră nodul deja existent, iar în caz negativ se construieşte un nod nou.

Pentru o linie de forma (2), se şterge expresia cu numărul *număr*. Se şterg nodurile corespunzătoare operatorilor, precum şi referinţele spre nodurile operanzilor. Dacă spre un operand nu mai indică nici o referinţă atunci se şterge şi nodul.

Linia (3) afişează expresia cu numărul *număr*.

Linia (4) evaluează expresia cu numărul *număr*, iar linia (5) asociază unui identificator o valoare.

16. Să se realizeze un program pentru traducerea unui text dintr-o limbă în alta. În plus, programul are facilitatea de a înlocui un cuvânt printr-un sinonim. Programul citeşte dintr-un fişier linii de forma:

cuvânt, echivalent în limba străină, sinonim, sinonim, ...

Secvenţa de linii se încheie cu **EOF**. În continuare, se citeşte un text care încheie tot cu **EOF**. Textul este parcurs cuvânt cu cuvânt şi pentru fiecare cuvânt se afişează lista sinonimelor sale. Utilizatorul poate selecta sinonimul dorit, care este înlocuit automat în textul iniţial. Textul astfel obţinut este tradus automat. Dacă nu există un echivalent străin pentru un cuvânt atunci se afişează un mesaj de eroare la care utilizatorul răspunde cu:

A - cuvânt străin

dacă cuvântul necunoscut este corect.

C - cuvânt corect

dacă cuvântul necunoscut este incorect.

Textul tradus este afişat.

17. Dintr-un fişier se citesc linii având forma:

identificator identificator număr identificator număr .....

Secvenţa de linii se încheie cu o linie vidă. Fiecare linie introduce o unitate de masură nouă, precum şi subunităţile ei şi relaţiile de conversie între acestea şi unitatea de bază. Numerele au forma: *secvenţă de cifre . secvenţă de cifre*.

Exemplu de linie:

m cm 0.01 mm 0.001

Aceeaşi unitate de măsură poate fi definită prin mai multe linii, dar toate unităţile şi subunităţile se reţin o singură dată. După citirea liniilor, programul răspunde interactiv la următoarele comenzi:

(1) *identificator <= număr unitate de masură*

Comanda asociază unui identificator o valoare şi o unitate de masură. În cazul în care identificatorul are deja o valoare asociată şi dacă unităţile aparţin aceleiaşi unităţi de bază, după ce s-a dat un mesaj de atenţionare, se reţine pentru identificator ultima valoare citită. În caz contrar, se dă un mesaj de eroare şi comanda se încheie.

.

(2) *identificator converteşte unitate de măsură*

Converteşte valoarea asociată identificatorului în unitatea specificată.

(3) *identificator tipăreste pi pf*

pi şi *pf* sunt două numere întregi. Afişează valoarea asociată identificatorului cu *pi* cifre la partea întreagă şi *pf* cifre pentru partea zecimală.

De exemplu, dacă *a* este 1305 cm, comanda:

a tipăreşte 1 2

va afişa: *a = 1.30 E3 cm*.

(4) *şterge valoare unitate de măsură*

Şterge din evidenţă toţi identificatorii care au o valoare mai mică decât cea din linia de comandă, iar unităţile aparţin aceleiaşi unităţi de bază.

(5) *afişează alfabetic*

Afişează identificatorii din evidenţă în următoarea formă:

unitate de bază:identificatorii cu unităţi de măsură aparţinând unităţii de bază.

Identificatorii sunt afişaţi ordonat alfabetic.

18. Să se scrie un program care rezolvă următoarea problemă. Se citesc dintr-un fişier linii de forma:

identificator = număr

număr are o parte întreagă, care este eventual urmată de o parte zecimală. Partea întreagă şi cea zecimală sunt secvenţe de cifre care sunt separate prin punct. Secvenţa de linii se încheie cu **EOF**. Fiecare identificator este reţinut într-o evidenţă împreună cu valoarea asociată lui. Dacă identificatorul apare deja în evidenţă, de fiecare dată se reţine şi noua lui valoare. În continuare, programul răspunde interactiv la următoarele comenzi:

s număr

număr are forma prezentată anterior. Se elimină din evidenţă toţi identificatorii care au asociată valoarea din comandă.

talf

Afişează în ordine alfabetică toţi identificatorii şi pentru fiecare identificator valorile asociate lui.

c identificator
Caută identificatorul şi, dacă îl găseşte, afişează toate valorile ataşate lui.

tval
Afişează, în ordinea valorii, identificatorii care au valoarea respectivă.
Afişarea se face astfel: pentru fiecare valoare se scriu identificatorii care au valoarea respectivă.

e
Termină programul.

19. Se citeşte o secvenţă de reguli (o regulă pe linie) cu forma:
identificator = identificator; identificator; ....
Secvenţa de linii se încheie cu **EOF**. Prin aceste reguli se asociază fiecărui identificator din stânga semnului = o secvenţă de identificatori. Un identificator poate fi definit prin mai multe reguli. Regulile sunt memorate într-o evidenţă. În continuare programul răspunde interactiv la următoarele comenzi:

a - Afişează ordonat, funcţie de identificatorul care apare în stânga lui =, toţi identificatorii şi secvenţele asociate lor.

c - Citeşte un cuvânt. Pentru acest cuvânt se afişează toţi identificatorii ataşaţi. În cazul în care nu există identificatori ataşaţi, programul afişează un mesaj de eroare corespunzător.

s - Citeşte un cuvânt, după care se şterg din evidenţă toate regulile pentru care cuvântul citit apare în partea stângă a semnului =.

f - Citeşte un cuvânt. Pentru cuvântul citit, pe baza regulilor ataşate, se formează un grup astfel: două reguli aparţin grupului dacă partea din dreapta semnului = începe cu cel puţin un identificator comun. Acest grup este afişat împreună cu partea comună.

e - Termină programul.